ObjectARX2006 实例教程

(ObjectARXLabs.chm)

PLgis 译编

二〇〇六年一月六日

目　　录

ObjectARX2006 实例教程

(ObjectARXLabs.chm)

PLgis 译编

开始前的准备

ObjectARX2006 适用的操作系统是 Windows 2000 或 Windows XP，在开始使用 ObjectARX2006 编程以前，要安装以下软件：

- AutoCAD2006
- ObjectARX2006
- Microsoft Visual C++ .NET 2002
- ObjectARX Wizard.

AutoCAD2006 和 Microsoft Visual C++ .NET 2002 的安装不必说了。

ObjectARX2006 的安装就是把 ObjectARX SDK 解压到你喜欢的位置，我们把它放在“C:\Program Files\Autodesk\ObjectARX2006\”中。

ObjectARX Wizard 的安装是在 Windows 下双击载入“<ObjectARX SDK 文件夹>\utils\ObjARXWiz\ ArxWizards.msi”即可。

实例 1.创建项目和注册 Hello 命令

本节目标

- 创建一个 ObjectARX 项目
- 设置编译器
- 设置链接器
- 创建 cpp 文件并添加代码
- 创建 def 文件并添加代码
- 编译和运行新项目

1.1 创建一个 ObjectARX 项目

运行 Visual C++ .NET，然后逐步创建第一个 ObjectARX 应用：

1.从 Visual C++ .NET 的菜单，选择[文件] >[新建]>[项目…]>[VC++ 项目]>[Win32 项目];

2.键入新项目的存放路径和项目名称，如“d:”，“Step01”;（图 1.1.2）

3.单击[完成]，弹出[Win 32 应用程序向导]对话框;

4.选择[应用程序设置]属性页，在“应用程序类型:”中选择 "DLL";（图 1.1.4）

5.单击[完成]，即完成了一个新项目的创建。

1.2 设置编译器

1. 从 Visual C++ .NET 的菜单，选择[视图]>[属性页]，弹出“属性页”对话框；
2. 在[配置]下拉列表中，选择“所有配置”；(图 1.2)
3. 选择[C/C++]节点，进行如下设置：
   [常规]/[附加包含目录]设置为：C:\Program Files\Autodesk\ObjectARX2006\inc
   [常规]/[警告等级]设置为：1 级 (/W1)
   [常规]/[检测 64 位可移植性问题]设置为：否
   [代码生成]/[运行时库]设置为：多线程 DLL (/MD)
4. 单击[应用]，完成编译器的设置。

1.3 设置链接器

1. 选择[链接器]节点，进行如下设置：
   [常规]/[输出文件]设置为：$(OutDir)/Step01.arx
   [常规]/[附加库目录]设置为：C:\Program Files\Autodesk\ObjectARX2006\lib
   [输入]/[附加依赖项]设置为：rxapi.lib acdb16.lib acge16.lib acad.lib acedapi.lib
2. 单击[完成]，完成链接器的设置。

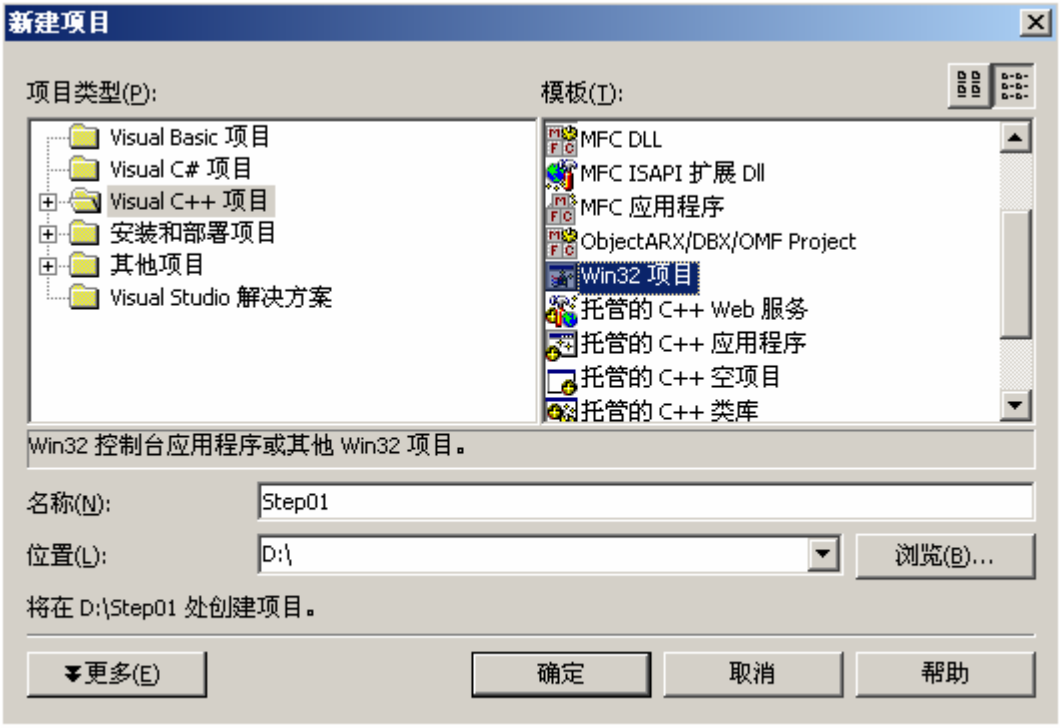


图1.1.2 新建一个Win32项目

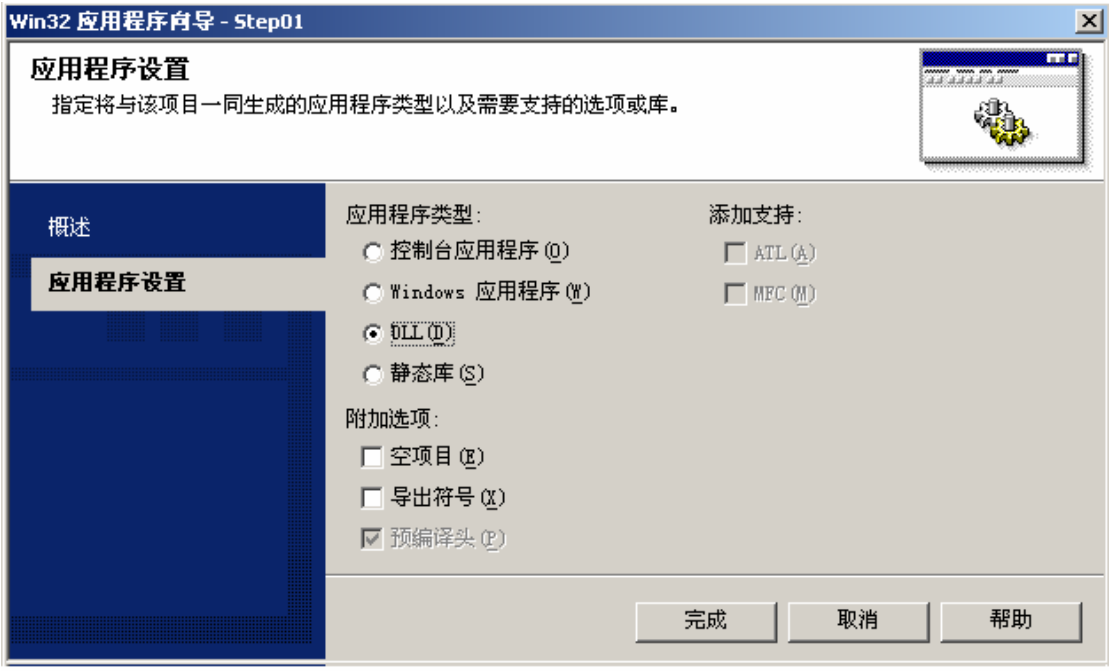


图1.1.4在"应用程序类型"中选择"DLL"

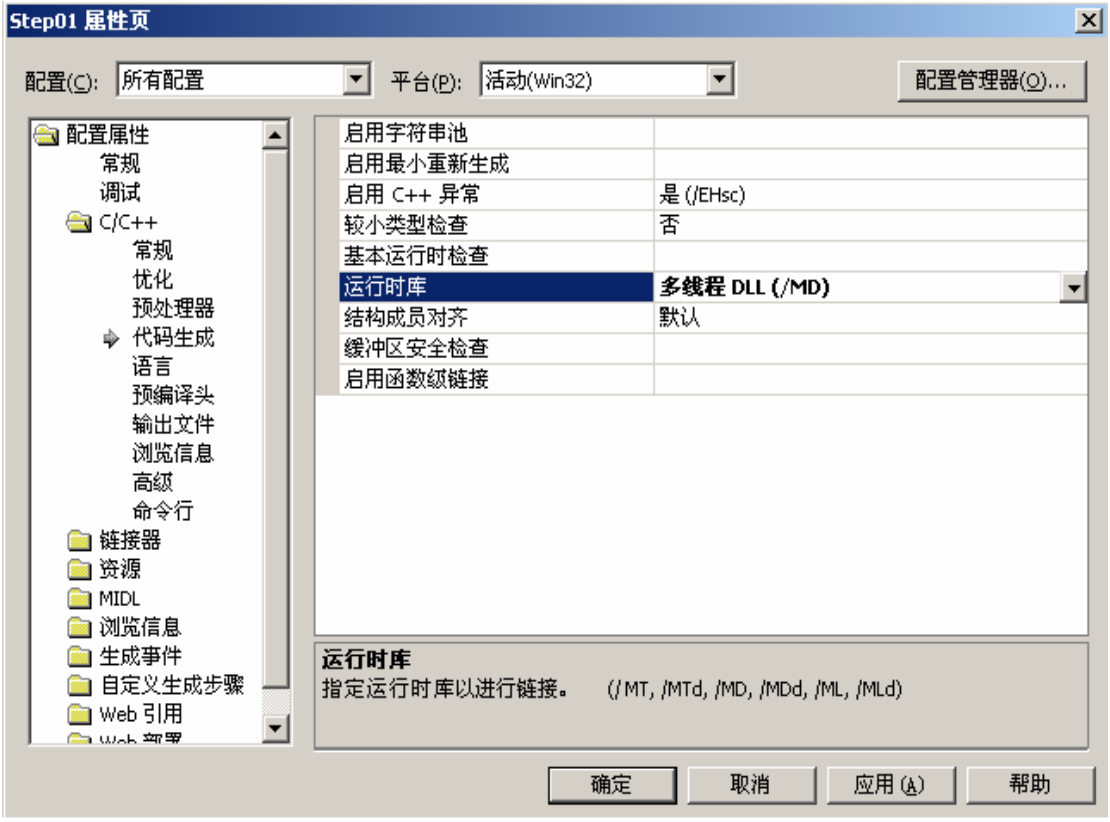


图1.2设置编译器

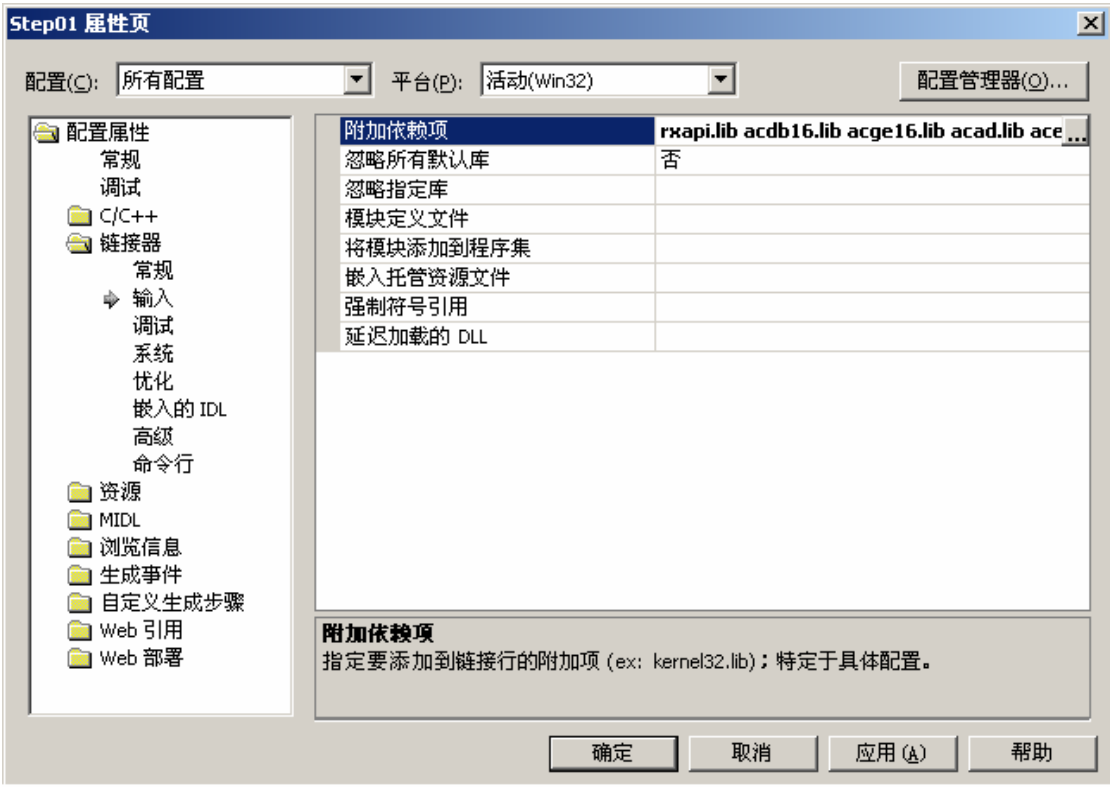


图1.3设置链接器

1.4 添加代码

1.4.1 创建一个新的 cpp 文件

(1)从 Visual C++ .NET 的菜单，选择[项目] >[添加新项]，弹出“添加新项”对话框；
(2)在[添加新项]下拉列表中，选择“C++ 文件 (.cpp)”；
(3)键入存放路径“d:\Step01\”和文件名“HelloWorld”；
(4)单击[打开]，创建了一个空的 cpp 文件。

1.4.2 添加代码

在 HelloWorld.cpp 窗口，添加如下代码：

```
#include "stdafx.h"
#include <aced.h>
#include <rxregsvc.h>

void initApp();
void unloadApp();
void helloWorld();

void initApp()
{
    // register a command with the AutoCAD command mechanism
```

```
    acedRegCmds->addCommand("HELLOWORLD_COMMANDS",
                            "Hello",
                            "Bonjour",
                            ACRX_CMD_TRANSPARENT,
                            helloWorld);
}

void unloadApp()
{
    acedRegCmds->removeGroup("HELLOWORLD_COMMANDS");
}

void helloWorld()
{
    acutPrintf("\nHello World!"); //实现本程序的具体功能
}

extern "C" AcRx::AppRetCode
acrxEntryPoint(AcRx::AppMsgCode msg, void* pkt)
{
    switch (msg)
    {
        case AcRx::kInitAppMsg:
            acrxDynamicLinker->unlockApplication(pkt);
            acrxRegisterAppMDIAware(pkt);
            initApp();
            break;
        case AcRx::kUnloadAppMsg:
            unloadApp();
            break;
        default:
            break;
    }
    return AcRx::kRetOK;
}
```

为了使 AutoCAD 能访问 acrxEntryPoint()函数，还需要创建一个 def 文件。

1.5 创建 DEF 文件

1.5.1 创建一个新的 def 文件

(1)从 Visual C++ .NET 的菜单，选择[项目] >[添加新项]，弹出“添加新项”对话框；

(2)在[添加新项]下拉列表中，选择“def 文件 (.def)”；

(3)键入存放路径“d:\Step01\”和文件名“ArxProject”；

(4)单击[打开]，创建了一个 def 文件。

1.5.2 添加代码

在 ArxProject.def 窗口，已经有了一行代码

LIBRARY Step01

再添加如下代码：

```
EXPORTS
        acrxEntryPoint          PRIVATE
        acrxGetApiVersion       PRIVATE
```

1.6 编译和运行 Step01

编译 Step01 项目，应该不会有什么问题，生成 Step01.arx。

运行 Acad2006，加载 Step01.arx。

在命令行键入“hello”。

则在命令行上打印输出：“Hello World!”。

至此，这个新项目的创建和测试完成。

实例 2.用 ObjectARX 向导创建项目

本节我们将使用 ObjectARX 向导来创建一个项目，了解如何用 ObjectARX 的函数来获得用户的输入。

本节目标

安装和使用 ObjectARX2006

- 安装 ObjectARX 向导
- 用向导创建 ObjectARX 项目
- 用 ObjectARX 向导添加命令
- 了解 ObjectARX 向导的 Autodesk 类管理器

演示 ObjectARX 的输入函数

执行“MyInput”命令，以演示 ObjectARX 的输入函数。

- 要求用户输入一个正整数。
- 要求用户输入一个实数，同时允许用户键入："PI" "A" "B" "C"。PI 对应 3.14,A 对应 10.0，B 对应 11.0，C 对应 12.0。
- 在命令执行过程中，适当处理用户按下了<ESC>键。
- 显示命令的提示。

演示 ObjectARX 的实体选择函数

执行“MySelect”命令，以演示 ObjectARX 的实体选择函数。

- a.要求用户选择一个实体。
- b.如果成功地选择了一个实体，显示选中对象的 ID

2.1 安装 ObjectARX 向导

ObjectARX 向导在 ObjectARX SDK 安装路径中：

<ObjectARX SDK folder>\utils\ObjARXWiz\ArxWizards.msi

在 Windows 下双击 ArxWizards.msi，即进入 ObjectARX 向导的安装过程。在安装了 ObjectARX 向导后，运行 VC++ .NET IDE，将在界面上出现"ObjectARX Addin" 工具条。

图2.1 ObjectARX工具条

2.2 用向导创建 ObjectARX 项目

1.从 Visual C++ .NET 的菜单，选择[文件] >[新建]>[项目…]>[VC++ 项目]>[ObjectARX/DBX/OMF Project];

2.键入新项目的存放路径和项目名称，如“d:”，“Step02”;

3.单击[完成]，弹出[ObjectARX/DBX/OMF Application Wizard for AutoCAD]对话框;

4.在[Your Registered Developer Symbol：]编辑框中，键入你认为能代表自己的字符串;（这里，我们键入“Asdk”）

5.单击[完成]，则向导会为我们创建一个新项目。

2.3 用 ObjectARX 向导添加命令

单击[ObjectARX Addin]工具条的"ObjectARX Commands"按钮，弹出“ObjectARX Commands”对话框。（图 2.3）

在对话框的“ARX command list”窗口中按下右键，在鼠标右键快捷菜单中选择 “New”，则自动添加了一个新的默认命令。把“Localized Name”改为“myInput”，把“"International Name”改为“_myInput”。用类似的方法添加"mySelect"命令。

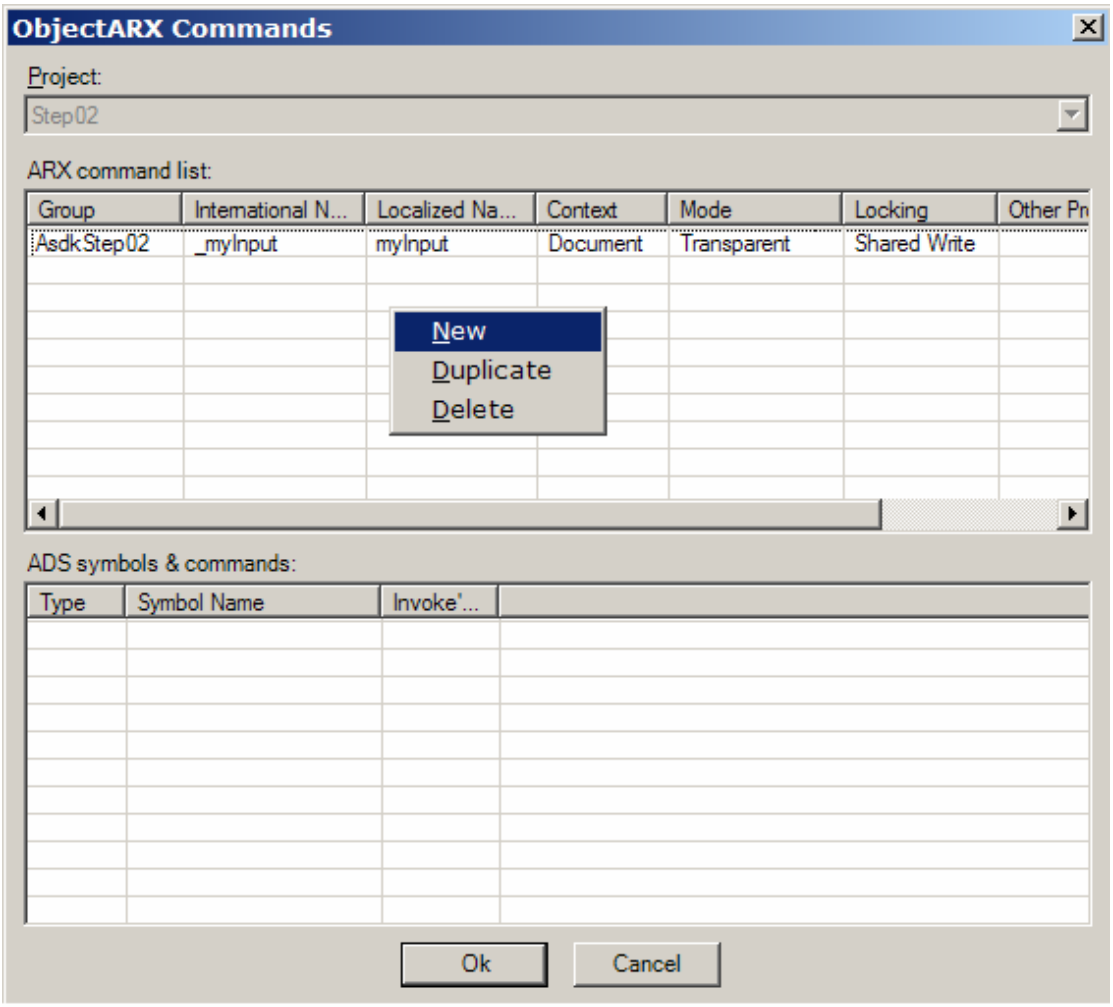


图2.3 用ObjectARX向导添加命令

单击[ok]，ObjectARX 向导将给 acrxEntryPoint.cpp 文件添加必须的代码。ObjectARX 向导还生成用一个名字是由命令组、下划线“_”和命令名组成的函数。例如，“myInput”命令对应的函数是：

AsdkStep02_myInput()

编程者可以在这些函数中，添写具体的代码。下面是 ObjectARX 向导生成的代码：

```
// (C) Copyright 2002-2005 by Autodesk, Inc.
//
// Permission to use, copy, modify, and distribute this software in
// object code form for any purpose and without fee is hereby granted,
// provided that the above copyright notice appears in all copies and
// that both that copyright notice and the limited warranty and
```

```
// restricted rights notice below appear in all supporting
// documentation.
//
// AUTODESK PROVIDES THIS PROGRAM "AS IS" AND WITH ALL FAULTS.
// AUTODESK SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTY OF
// MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE.    AUTODESK, INC.
// DOES NOT WARRANT THAT THE OPERATION OF THE PROGRAM WILL BE
// UNINTERRUPTED OR ERROR FREE.
//
// Use, duplication, or disclosure by the U.S. Government is subject to
// restrictions set forth in FAR 52.227-19 (Commercial Computer
// Software - Restricted Rights) and DFAR 252.227-7013(c)(1)(ii)
// (Rights in Technical Data and Computer Software), as applicable.
//

//-----------------------------------------------------------------------------
//----- acrxEntryPoint.h
//-----------------------------------------------------------------------------
#include "StdAfx.h"
#include "resource.h"

//-----------------------------------------------------------------------------
#define szRDS _RXST("Asdk")

//-----------------------------------------------------------------------------
//----- ObjectARX EntryPoint
class CStep02App : public AcRxArxApp {

public:
	CStep02App () : AcRxArxApp () {}

	virtual AcRx::AppRetCode On_kInitAppMsg (void *pkt) {
		// TODO: Load dependencies here

		// You *must* call On_kInitAppMsg here
		AcRx::AppRetCode retCode =AcRxArxApp::On_kInitAppMsg (pkt) ;

		// TODO: Add your initialization code here

		return (retCode) ;
	}

	virtual AcRx::AppRetCode On_kUnloadAppMsg (void *pkt) {
		// TODO: Add your code here
```

```
        // You *must* call On_kUnloadAppMsg here
        AcRx::AppRetCode retCode =AcRxArxApp::On_kUnloadAppMsg (pkt) ;

        // TODO: Unload dependencies here

        return (retCode) ;
    }

    virtual void RegisterServerComponents () {
    }

    // - AsdkStep02._myInput command (do not rename)
    static void AsdkStep02_myInput(void)
    {
        // Add your code for command AsdkStep02._myInput here
    }

    // - AsdkStep02._mySelect command (do not rename)
    static void AsdkStep02_mySelect(void)
    {
        // Add your code for command AsdkStep02._mySelect here
    }
} ;

//-----------------------------------------------------------------------------
IMPLEMENT_ARX_ENTRYPOINT(CStep02App)

ACED_ARXCOMMAND_ENTRY_AUTO(CStep02App, AsdkStep02, _myInput, myInput,
ACRX_CMD_TRANSPARENT, NULL)
ACED_ARXCOMMAND_ENTRY_AUTO(CStep02App, AsdkStep02, _mySelect, mySelect,
ACRX_CMD_TRANSPARENT, NULL)
```

2.4 ObjectARX 向导的 Autodesk 类管理器

让我们了解 ObjectARX 向导最重要的特征，Autodesk 类管理器 ACE。

单击[ObjectARX Addin]工具条的第 2 个按钮，弹出“Autodesk Class Explorer”窗口。

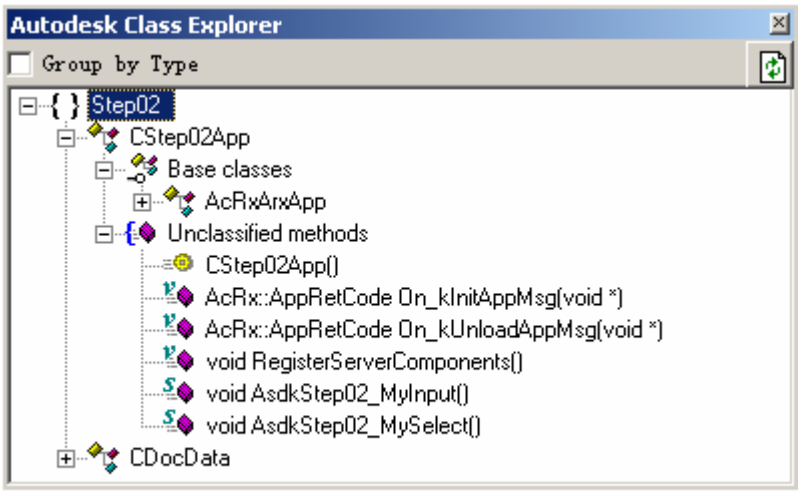


图2.4 ObjectARX向导的类管理器窗口

ACE 显示的类结构树和 VC7.0 的类管理器类似。ACE 不但显示类的结构树，还提供了直接访问 ObjectARX 向导的中枢。

在创建了 ARX/DBX 项目并调用 ACE 后，一旦 ACE 的类结构树被载入，ACE 就会有相应的鼠标右键菜单。

使用 ACE 的右键菜单：

右击项目节点，出现允许调用各种向导的菜单。这些向导允许你添加反应器、定制实体和 MFC 扩展类等等。

选中"Group by Type"可选项，ACE 将显示类的方法。这些方法按类编排显示，类似 ObjectARX 的在线帮助。ACE 显示的各种类别的信息存放在"classmethodgroups.xml"文件中，该文件位于 ObjectARX Wizard 文件夹中。实际上，你也可以把自己最常用的类添加近来。

在派生类中，也可以用执行基类的函数。

2.5 ObjectARX 的头文件

要使用各种 ObjectARX 类,在你的.cpp 源文件或.h 头文件中就必须包含相应的 ObjectARX 头文件。有效的 ObjectARX 头文件位于 ObjectARX SDK 安装文件夹中：

<ObjectARX SDK folder>\inc\

很幸运，ObjectARX 向导在创建 ObjectARX 项目时，在 StdAfx.h 文件中有下列包含语句：

```
#include "arxHeaders.h"
```

而 arxheaders.h 中有在<ObjectARX SDK folder>\inc\中所有的 ObjectARX 头文件。所以，我们一般不用再填写包含语句。

2.6 用 ObjectARX 获得输入

2.6.1 ObjectARX 提供的输入函数：

acedGetInt　用于获得一个整数值
acedGetReal　用于获得一个实数值
acedGetString　用于获得一个字符串
acedGetAngle　用于获得一个角度值
acedGetKword　用于获得一个关键字
acedInitGet　用于初始化acedGetXXXX函数

acedGetFileD　用于从文件对话框获得一个实型值返回文件集
acedGetPoint　用于拾取一个点
acedGetDist　用于得到两点的距离
acedGetCorner　用拖动方式获得对角点

2.6.2 ObjectARX 提供的实体选择函数：

acedEntSel　用于选择一个实体
acedNEntSel　用于选择一个嵌套实体
acedNEntSelP　用于选择一个嵌套实体
acutSSGet　用于选择多个实体

2.7 给命令添加代码

在用 ObjectARX 向导添加命令时，向导已经在 acrxEntryPoint.cpp 文件中自动创建了 myInput 和 mySelect 函数。但他们只是两个函数的框架，函数要完成的具体功能，还需要我们在函数中添加适当的代码，才能真正实现。

2.7.1 给 myInput 函数添加代码

在 myInput 函数的// Add your code for command AsdkStep02._myInput here 行的下面写入代码，使 myInput 函数如下所示：

```
// - AsdkStep02._myInput command (do not rename)
static void AsdkStep02_myInput(void)
{
	// Add your code for command AsdkStep02._myInput here
	int stat, iValue ;
	double rValue ;
	char kWord [133] ;

	acedInitGet (RSG_NONEG | RSG_NOZERO, "") ;

	if ( acedGetInt ("\nEnter an integer value: ", &iValue) != RTNORM )
		return ;

	rValue =12.0 ;
	acedInitGet (RSG_NOZERO, "PI A B C") ;
	stat =acedGetReal ("\nEnter a real value [PI/A/B/C] <C>: ", &rValue) ;
	switch (stat) {
		case RTCAN: // User termination
			return ;
		case RTKWORD: // Got a keyword
			if ( acedGetInput (kWord) != RTNORM )
				return ;
			if ( !strcmp ("PI", kWord) )
				rValue =3.14 ;
```

```
                else if ( !strcmp ("A", kWord) )
                        rValue =10 ;
                else if ( !strcmp ("B", kWord) )
                        rValue =11 ;
                else if ( !strcmp ("C", kWord) )
                        rValue =12 ;
                break ;
        case RTNONE:
                acutPrintf ("\nTake default rValue %lf", rValue) ;
                break ;
        case RTNORM:
                break ;
        default:
                return ;
    }

    acutPrintf ("\nInteger : %d", iValue) ;
    acutPrintf ("\nReal    : %lf", rValue) ;
}
```

2.7.2 给 mySelect 函数添加代码

在 mySelect 函数的// Add your code for command AsdkStep02._mySelect here 行的下面写入代码，使 mySelect 函数如下所示：

```
// - AsdkStep02._mySelect command (do not rename)
static void AsdkStep02_mySelect(void)
{
    // Add your code for command AsdkStep02._mySelect here
    ads_name en ;
    ads_point pt ;

    if ( acedEntSel (NULL, en, pt) != RTNORM )
        return ;

    acutPrintf ("\nYou selected entity ID: %ld", en [0]) ;
}
```

2.8 编译和执行命令

编译 Step02 项目，应该不会有什么问题，生成 Step02.arx。

运行 Acad2006，加载 Step02.arx。

在命令行键入“myinput”，执行过程如下：

```
命令: myinput
Enter an integer value: -3
值必须为 正且非零。
Enter an integer value: 2.2
需要正的非零整数.
Enter an integer value: 3
Enter a real value [PI/A/B/C] <C>: b
Integer : 3
Real       : 11.000000
命令:
```

在命令行键入“myselect”,执行过程如下：

```
命令: mySelect
选择对象:
You selected entity    ID: 2130063240
命令:
```

至此，这个新项目的创建和测试完成。

实例 3.符号表

本节将了解 AutoCAD 的符号表。我们将创建一个新层、一个块定义和说明如何使用遍历器遍历模型空间块表记录中的实体。本节还要介绍对象的打开以便处理读写操作。

注意:任何为了读写而打开的对象,在结束读写操作后必须将其关闭,否则将引起 AutoCAD 错误。

本节目标

在本练习中，我们将创建“Create”和“SetLayer”两个 AutoCAD 命令。

在运行“Create”命令时，该命令将:

- 创建一个名为“USER”新层和创建一个名为“Employee”的新块的定义。
- 在创建新层过程中，要向 AcDbLayerTable 中添加 AcDbLayerTableRecord。
- 在创建新块的定义过程中，要向 AcDbBlockTable 中添加 AcDbBlockTableRecord。为实现块定义，构成块定义的实体要添加到 AcDbBlockTableRecord 中。用 AutoCAD 的“Insert”命令创建块定义的一个实例。一个块定义的实例叫做 AcDbBlockReference 实体。AcDbBlockReference 实体应该添加到模型空间或某个 AutoCAD 图纸空间布局中。

在运行“Setlayer”命令时，该命令将:

- 在模型空间搜索“Employee”块的实例（AcDbBlockReference 实体参照"EMPLOYEE" AcDbBlockTableRecord)。将找到的每个"EMPLOYEE"块的图层改变到“USER”层。在模型空间搜索"EMPLOYEE"块的实例要利用块表遍历器 AcDbBlockTableRecordIterator。
- "EMPLOYEE"块的定义的模样是个由 3 个圆和一个圆弧组成的“笑脸”。

首先,与“输入函数”一章类似,创建创建一个新的 ObjectARX 项目 Step03,并用 ObjectARX 向导添加“Create”和“SetLayer”两个命令。

3.1 创建应用函数的头文件和执行文件

参照 2.3 节的步骤，用向导创建 ObjectARX 项目“Step03”。

参照 2.4 节的步骤，用 ObjectARX 向导添加 myCreate 和 mySetlayer 命令。

在本练习中，我们将实现在后面的命令中要用到的应用函数。我们要编写 createLayer()和 createBlockRecord()两个函数用于实现“Create”命令。这两个函数放在 utilities.cpp 文件中，函数的声明放在 utilities.h 文件中。

在项目的解决方案资源管理器中，右键 Step03 节点，在弹出的快捷菜单中，选择[增加]>[增加新项]>，创建 utilities.h 文件和 utilities.cpp 文件。

在头文件 utilities.h 中，添加下列代码:

```
#include "StdAfx.h"
```

```
//Declaration of createLayer():
Acad::ErrorStatus createLayer (const char* layerName, AcDbObjectId& layerId);

//Declaration of createBlockRecord():
Acad::ErrorStatus createBlockRecord (const char *name);
```

在文件 utilities.cpp 中，首先添加一行包含语句：

```
#include "utilities.h"
```

3.1.1 自定义 createLayer()函数的实现

添加下列函数说明：

```
//
// Create a new layer or return the ObjectId if it already exists
//
// In :
// const char* layerName : layer name
// Out :
// AcDbObjectId& layerId : ObjectId of the created or existing layer
//
```

3.1.2 createLayer()函数的算法

1.从当前工作空间(AcDbLayerTable, AcDbDatabase::getLayerTable())获得层表。用 acdbHostApplicationServices()->workingDatabase()获得当前工作数据库。

2.检查同名的层是否已存在(AcDbLayerTable::getAt())。如果已存在，用 createLayer()函数的第 2 段代码，获得并返回该对象的 ID。

3. 如果不存在，就用“new”操作创建一个新层(AcDbLayerTableRecord)。因为还要修改层表，故层表必须打开。把新创建的 Layer Table Record 用 AcDbLayerTable::add()加到层表中。在关闭了新层后，返回它的 ID。

4.在使用 AcDbLayerTable, AcDbLayerTableRecord 时，需要包含<dbsymtb.h>。因为 arxHeaders.h 包含所有 ObjectARX 的头文件，stdafx.h 中有包含 arxHeaders.h 的语句，而 utilities.h 中包含 stdafx.h，所以我们不必添加<dbsymtb.h>这个头文件。

3.1.3 createLayer()函数的代码

```
Acad::ErrorStatus createLayer (const char *layerName, AcDbObjectId &layerId) {
    Acad::ErrorStatus es ;
    AcDbLayerTable *pTable ;

    layerId =AcDbObjectId::kNull ;
    if ( (es =acdbHostApplicationServices ()->workingDatabase ()->getLayerTable (pTable, AcDb::kForRead)) ==
Acad::eOk ) {
        // Use the overload of AcDbLayerTable::getAt() that returns the id
        if ( (es =pTable->getAt (layerName, layerId, Adesk::kFalse)) != Acad::eOk ) {
            // Create a new layer table record using the layer name passed in
```

```
                AcDbLayerTableRecord *pLTRec =new AcDbLayerTableRecord ;
                pLTRec->setName (layerName) ;
                // Set other properties - color, linetype, state - if desired
                // this will require more input than this simple example provides
                if ( (es =pTable->upgradeOpen ()) == Acad::eOk ) {
                    es =pTable->add (layerId, pLTRec) ;
                    // Since the new layer was successfully added to the database,
                    // close it - DON'T delete it
                    pLTRec->close () ;
                } else {
                    // The object is not part of the database, therefore we should
                    // delete it to avoid a memory leak
                    delete pLTRec ;
                }
            }
            pTable->close () ;
        }
        return (es) ;
}
```

3.1.4 自定义 createBlockRecord()函数的实现

添加下列函数说明：

```
//
// Create a new layer or return the ObjectId if it already exists
//
// In :
// const char* layerName : layer name
// Out :
// AcDbObjectId& layerId : ObjectId of the created or existing layer
//
```

3.1.5 createBlockRecord()函数的算法

1.从当前工作数据库(AcDbBlockTable, AcDbDatabase::getBlockTable())获得块表。不要忘记关闭块表。

2.检查是否已存在块表记录(AcDbBlockTable::has())。如果已存在，返回错误状态(Acad::eDuplicateKey)。

3.用“new”操作创建一个新的块表记录并把它加到块表中(AcDbBlockTableRecord, AcDbBlockTable::add())。因为要修改块表，故必须将其打开。初始化新块表记录。把原点设置为(0,0,) (AcDbBlockTableRecord::setOrigin (AcGePoint3d::kOrigin))。设置块表名称。如果把块表记录成功地加入 AutoCAD 数据库，则关闭块表。

4.创建“Employee”实体并添加到新的块表记录中。(AcDbBlockTableRecord::appendAcDbEntity())“Employee”实体由 3 个圆(AcDbCircle)和一个圆弧(AcDbArc)构成，其特征见下面的表格。将脸置为黄色，眼睛置为蓝色，嘴置为红色。

(AcDbEntity::setColorIndex())

5.别忘记关闭新创建的实体、块表记录和块表。

6.在使用 AcDbCircle, AcDbArc 时，需要包含<dbents.h>文件。但我们同样不必添加这个包含语句。

7.PI 可以定义为 3.141592654.。

3.1.6 EMPLOYEE 块的参数定义

块名 EMPLOYEE	AcDbBlock Reference				
块原点	(0, 0, 0)				
脸的特征	AcDbCircle (yellow)	中心= (0,0,0)	半径=1.0		
左眼	AcDbCircle (blue)	中心= (0.33,0.25 ,0)	半径=0.1		
右眼	AcDbCircle (blue)	中心= (-0.33,0.2 5,0)	半径=0.1		
嘴	AcDbArc (red)	中心= (0,0.5,0)	半径=1.0	起始角度 =PI+(PI*0.3)	终止角度 =PI+(PI*0.7)

3.1.7 createBlockRecord()函数的代码

```
//
// Create a new block table record and add the entities of the employee to it
//
// In :
//        const char* name : Name of block table record
//

Acad::ErrorStatus createBlockRecord (const char *name) {
        // First, check if a block of the same name already exists
        // by verifying in the current database block table.
        AcDbBlockTable *pBlockTable ;
        // Open the block table for read
        Acad::ErrorStatus es ;
        if ( (es =acdbHostApplicationServices ()->workingDatabase ()->getBlockTable (pBlockTable,
AcDb::kForRead)) != Acad::eOk )
                return (es) ;

        if ( pBlockTable->has (name) == Adesk::kTrue ) {
                pBlockTable->close () ;
                return (Acad::eDuplicateKey) ;
        }
        // Now we know the block does not exist, so we create it
```

```
    // using the name passed in.
    AcDbBlockTableRecord *pBlockTableRecord =new AcDbBlockTableRecord ;
    pBlockTableRecord->setName (name) ;
    // To keep it simple, we use the origin for the insertion point
    pBlockTableRecord->setOrigin (AcGePoint3d::kOrigin) ;
    // Open the block table for write
    // since we are adding a new block definition
    if ( (es =pBlockTable->upgradeOpen ()) != Acad::eOk ) {
        delete pBlockTableRecord ;
        pBlockTable->close () ;
        return (es) ;
    }
    // Add the new block table record to the block table.
    // For now, the block table record is empty.
    if ( (es =pBlockTable->add (pBlockTableRecord)) != Acad::eOk ) {
        // The block table record has not been added
        // to the block table, so we have to delete it.
        pBlockTable->close();
        delete pBlockTableRecord;
        return (es) ;
    }
    pBlockTable->close () ;
    // Now the block table record is in the database, but is empty
    // (has no sub-entity).
    // Note that after having been added to the database, an object or an entity
    // is implicitely opened for write.
    //
    // So we create the sub entities to append to the block
    // which will represent a "happy face":
    // the block should consist of a round yellow face (circle)
    // two blue eyes (circles) and a red mouth (arc)
    AcDbCircle *pFace =new AcDbCircle (AcGePoint3d::kOrigin, AcGeVector3d::kZAxis, 1.0) ;
    AcDbCircle *pLeftEye =new AcDbCircle (AcGePoint3d (0.33, 0.25, 0.0), AcGeVector3d::kZAxis, 0.1) ;
    AcDbCircle *pRightEye =new AcDbCircle (AcGePoint3d (-0.33, 0.25, 0.0), AcGeVector3d::kZAxis, 0.1) ;
    AcDbArc *pMouth =new AcDbArc (AcGePoint3d (0, 0.5, 0), 1.0, 3.141592 + (3.141592 * 0.3), 3.141592 +
(3.141592 * 0.7)) ;
    // Set the color property.
    pFace->setColorIndex (2) ;
    pLeftEye->setColorIndex (5) ;
    pRightEye->setColorIndex (5) ;
    pMouth->setColorIndex (1) ;
    // add the entities to the new block table record
    if ( (es =pBlockTableRecord->appendAcDbEntity (pFace)) != Acad::eOk ) {
        delete pFace ;
```

```
            delete pLeftEye ;
            delete pRightEye ;
            delete pMouth ;
            pBlockTableRecord->erase () ;
            pBlockTableRecord->close () ;
            return (es) ;
        }
        pFace->close () ;

        if ( (es =pBlockTableRecord->appendAcDbEntity (pLeftEye)) != Acad::eOk ) {
            delete pLeftEye ;
            delete pRightEye ;
            delete pMouth ;
            pBlockTableRecord->erase () ;
            pBlockTableRecord->close () ;
            return (es) ;
        }
        pLeftEye->close () ;

        if ( (es =pBlockTableRecord->appendAcDbEntity (pRightEye)) != Acad::eOk ) {
            delete pRightEye ;
            delete pMouth ;
            pBlockTableRecord->erase () ;
            pBlockTableRecord->close () ;
            return (es) ;
        }
        pRightEye->close () ;

        if ( (es =pBlockTableRecord->appendAcDbEntity (pMouth)) != Acad::eOk ) {
            delete pMouth ;
            pBlockTableRecord->erase () ;
            pBlockTableRecord->close () ;
            return (es) ;
        }
        pMouth->close () ;

        pBlockTableRecord->close () ;

        return (Acad::eOk) ;
}
```

3.2 实现 CREATE 和 SETLAYER 命令

为了使用应用函数，在 acrxEntryPoint.cpp 加入下面的语句：

```
#include "utilities.h"
```

3.2.1 CREATE 命令的算法

ObjectARX 向导会在 acrxEntryPoint.cpp 中自动创建将在 CREATE 命令执行时调用的 AsdkStep03_CREATE()函数。在 AsdkStep03_CREATE()函数中实现下列功能。

1.以参数“USER”做为层名调用 createLayer()函数，以参数“EMPLOYEE”做为块名调用 createBlockRecord()函数。

2.在层表改变后，调用 ObjectARX 的全局函数 applyCurDwgLayerTableChanges()。这个函数会强制 AuoCAD 自我更新以在 AutoCAD 编辑器的当前绘图的层表记录中使用改变了的模式。

3.2.2 CREATE 命令的代码

```
static void AsdkStep03_CREATE(void)
{
    // Add your code for command AsdkStep03._myCreate here
    // TODO: Implement the command
    // Create a new layer named "USER"
    // createLayer returns the object ID of the newly created layer
    AcDbObjectId layerId ;
    if ( createLayer ("USER", layerId) != Acad::eOk ) {
        acutPrintf ("\nERROR: Couldn't create layer record.") ;
        return ;
    }
    // This is not always needed, but a call to 'applyCurDwgLayerTableChanges()'
    // will synchronize the newly created layer table change with the
    // rest of the current DWG database.
    applyCurDwgLayerTableChanges () ;

    acutPrintf ("\nLayer USER successfully created.") ;

    // Create a new block definition named "EMPLOYEE"
    if ( createBlockRecord ("EMPLOYEE") != Acad::eOk )
        acutPrintf ("\nERROR: Couldn't create block record.") ;
    else
        acutPrintf ("\nBlock EMPLOYEE successfully created.") ;
}
```

3.2.3 SETLAYER 命令的算法

ObjectARX 向导会在 acrxEntryPoint.cpp 中自动创建将在 SETLAYER 命令执行时调用的 AsdkStep03_ SETLAYER()函数。在 AsdkStep03_ SETLAYER()函数中实现下列功能。（改变“EMPLOYEE”块参考的层）

1.打开当前工作数据库的块表。

2.得到模型空间块表记录。(AcDbBlockTable::getAt(), ACDB_MODEL_SPACE)

3. 得到块表记录遍历器(AcDbBlockTableRecordIterator, AcDbBlockTableRecord::newIterator()) 以遍历模型空间块表记录。

4. 遍历模型空间 MODEL_SPACE (AcDbBlockTableRecordIterator::start(), AcDbBlockTableRecordIterator::done(), AcDbBlockTableRecordIterator::step()).

5.用 acdbOpenObject()以读写方式打开对象。

6.获得实体(AcDbBlockTableRecordIterator::getEntity())。

7.检查该实体是否是块参考(pEnt->isA() != AcDbBlockReference::desc())

8.获得该参考的块表记录(AcDbBlockReference::blockTableRecord())并检查该块表记录的名字是否是“EMPLOYEE”。

9.改变层(setLayer())。

10.别忘了关闭所有打开的对象并删除遍历器。

3.2.4 SETLAYER 命令的代码

```
static void AsdkStep03_SETLAYER(void)
{
    // Add your code for command AsdkStep03._mySetLayer here
    // TODO: Implement the command
    // Iterate through Model Space to find every instance of the EMPLOYEE block
    // When found, change its layer to "USER"
    Acad::ErrorStatus es ;
    AcDbBlockTable *pBlockTbl ;
    AcDbBlockTableRecord *pMS ;

    // Get the block table
    if ( (es =acdbHostApplicationServices ()->workingDatabase ()->getBlockTable (pBlockTbl,
AcDb::kForRead)) != Acad::eOk ) {
        acutPrintf ("\nCouldn't open the block table!") ;
        return ;
    }
    // Get the Model Space record and open it for read.
    if ( (es =pBlockTbl->getAt (ACDB_MODEL_SPACE, pMS, AcDb::kForWrite)) != Acad::eOk ) {
        acutPrintf ("\nCouldn't get Model Space! Drawing corrupt.\n") ;
        pBlockTbl->close () ;
        return ;
    }
    pBlockTbl->close () ;

    // Declare the appropriate iterator type
    // Get the iterator from the object to be iterated through

    // In this case, the Model Space block table record will provide the iterator
    // start at the beginning of the record and skip deleted entities
    AcDbBlockTableRecordIterator *pBtrIter ;
    if ( (es =pMS->newIterator (pBtrIter) ) != Acad::eOk ) {
```

```
            acutPrintf ("\nCouldn't create Model Space iterator.") ;
            pMS->close () ;
            return ;
        }

        char *blockName ;
        AcDbEntity *pEnt ;
        AcDbBlockTableRecord *pCurEntBlock ;
        AcDbObjectId blockId ;

        for ( pBtrIter->start (); !pBtrIter->done () ; pBtrIter->step () ) {
            // First open each entity for read, just to check its class
            // if it's what we want, we can upgrade open later
            // Don't bother with erased entities
            if ( (es =pBtrIter->getEntity (pEnt, AcDb::kForRead)) != Acad::eOk ) {
                acutPrintf ("\nCouldn't open entity.");
                continue ;
            }
            if ( pEnt->isA() != AcDbBlockReference::desc () ) {
                pEnt->close () ;
                continue ;
            }
            // Get the insert's block table record and compare its name
            // to make sure we've got the right block.    If so, set the layer
            blockId =(AcDbBlockReference::cast (pEnt))->blockTableRecord () ;
            if ( acdbOpenObject ((AcDbObject *&)pCurEntBlock, blockId, AcDb::kForRead) == Acad::eOk ) {
                pCurEntBlock->getName(blockName);
                if ( strcmp (blockName, "EMPLOYEE") == 0 ) {
                    if ( pEnt->upgradeOpen () == Acad::eOk )
                        // setLayer also has an overload that takes a layer ID
                        // but to avoid global variables we specify the layer name
                        pEnt->setLayer ("USER") ;
                }
                pCurEntBlock->close () ;
                acdbFree (blockName) ;
            }
            pEnt->close () ;
        }

        // delete, rather than close, the iterator object
        delete pBtrIter ;
        pMS->close () ;
}
```

3.3 测试 CREATE 和 SETLAYER 命令

在一个新图中执行 CREATE 命令，然后用 AutoCAD 的 INSERT 命令在 0 层增加一个或几个 EMPLOYEE 块的实例。接着再执行 SETLAYER 命令把 EMPLOYEE 块改变到 USER 层上。可以用 AutoCAD 的 LIST 命令检测结果。

```
命令: mycreate
Layer USER successfully created.
Block EMPLOYEE successfully created.
```

用 Acad 的 Insert 命令在 0 层插入一个 EMPLOYEE 块，接着可以用 Acad 的 List 命令查看该块的基本属性。

```
命令:
命令: list
找到 1 个
                    BLOCK REFERENCE   图层: 0
                              空间: 模型空间
                     句柄 = fa
                块名: "EMPLOYEE"
                   于 点，  X= 929.0598   Y= 583.2882   Z=    0.0000
       X 比例因子:    1.0000
       Y 比例因子:    1.0000
         旋转角度:    0
       Z 比例因子:    1.0000
  按统一比例缩放:    否
         允许分解:    是
```

下面执行 mySetLayer 命令：

```
命令: mysetlayer
```

接着可以用 Acad 的 List 命令查看该块的基本属性已经改变。

```
命令:
命令: list
找到 1 个
                    BLOCK REFERENCE   图层: USER
                              空间: 模型空间
                     句柄 = f9
                块名: "EMPLOYEE"
                   于 点，  X= 888.0560   Y= 576.8192   Z=    0.0000
       X 比例因子:    1.0000
       Y 比例因子:    1.0000
         旋转角度:    0
       Z 比例因子:    1.0000
  按统一比例缩放:    否
         允许分解:    是
```

实例 4.命名对象词典_Xrecords

本节将介绍词典的概念并对命名对象词典(NOD)进行研究。NOD 用于在图形文件中保存非图形数据。推荐用 NOD 创建一个基准词典(AcDbDictionary)，然后将需要保存的分类数据添加到自己的基准词典中。我们还将引入 Xrecord 对象(AcDbXrecord)，和现在仍然在使用的扩展实体数据相比，Xrecord 对象大大优化了数据的存储结构。

我们的基准词典对象有一个"ASDK_EMPLOYEE_DICTIONARY"关键字，代表雇员记录。我们同样在这个新的词典中建立空的 Xrecord 对象，对应雇员记录。雇员记录（关键字和关联的 AcDbXrecord）被加到 NOD 的"ASDK_EMPLOYEE_DICTIONARY"基准词典中。由于在词典中有了这些条目，使我们可以很容易地快速遍历数据库中的所有雇员。

本节目标

在本练习中，我们将创建“ADDENTRY”，“LISTENTRIES”和“REMOVEENTRY”三个 AutoCAD 命令。

在运行“ADDENTRY”命令时，该命令将：

- 检查在命名对象词典 NOD 中是否存在“ASDK_EMPLOYEE_DICTIONARY”基准词典 AcDbDictionary。如果不存在，“ADDENTRY”将在 NOD 中创建基准词典。
- 提示输入一个雇员名，然后把一个 AcDbXrecord 对象添加到我们的基准词典中。如果该雇员已经存在于“ASDK_EMPLOYEE_DICTIONARY”中，该请求将被忽略。

在运行“LISTENTRIES”命令时，该命令将：

- 检查在命名对象词典中是否存在“ASDK_EMPLOYEE_DICTIONARY”基准词典 AcDbDictionary。如果不存在，本命令终止。
- 使用 AcDbDictionaryIterator 遍历“ASDK_EMPLOYEE_DICTIONARY”词典，并打印出所有保存的雇员名。

在运行“REMOVEENTRY”命令时，该命令将：

- 提示输入一个雇员名。
- 检查在“ASDK_EMPLOYEE_DICTIONARY”基准词典中是否命名对象词典。如果不存在，本命令终止。
- 如果存在，删除该雇员条目。

下面是已准备的关联到命名对象词典的对象：

```
NOD (class:AcDbDictionary)
     |_"ASDK_EMPLOYEE_DICTIONARY" (class:AcDbDictionary)
               |_"Charles" (class:AcDbXrecord)
               |_"Cyrille" (class:AcDbXrecord)
               |_"Gabriel" (calss:AcDbXrecord)
               |_"Henry" (class:AcDbXrecord)
               |_"Peter" (class:AcDbXrecord)
               |_"Kean" (class:AcDbXrecord)
               |_"Thilak" (class:AcDbXrecord)
```

4.1 实现 ADDENTRY，LISTENTRIES 和 REMOVEENTRY 命令

首先，参照 2.3 节的步骤，用向导创建 ObjectARX 项目“Step04”。

然后参照 2.4 节的步骤，用 ObjectARX 向导创建 MyADDENTRY，MyLISTENTRIES 和 MyREMOVEENTRY 命令。

4.1.1 ADDENTRY 命令的算法

ObjectARX 向导会在 acrxEntryPoint.cpp 中自动创建将在 ADDENTRY 命令执行时调用的 AsdkStep04_ADDENTRY()函数。在 AsdkStep04_ ADDENTRY ()函数中实现下列功能。

1.提示用户输入雇员名(acedGetString())。

2. 在当前工作数据库中得到命名对象词典(AcDbDictionary, AcDbDatabase::getNamedObjectsDictionary())。

3. 检查在 NOD 中是否已有了″ASDK_EMPLOYEE_DICTIONARY″。(AcDbDictionary::getAt())

4. 如果不存在，用关键字"ASDK_EMPLOYEE_DICTIONARY"创建新的 AcDbDictionary，然后加到命名对象词典中。(new AcDbDictionary, AcDbDictionary::setAt())

5. 检查雇员名是否已经存在于"ASDK_EMPLOYEE_DICTIONARY"中。

6. 如果还没有该雇员词典，就创建一个新的 AcDbXrecord，然后加到"ASDK_EMPLOYEE_DICTIONARY" (AcDbDictionary::setAt())中。

7.不要忘记关闭命名对象词典、"ASDK_EMPLOYEE_DICTIONARY" 和 Xrecord。

8.使用 AcDbXrecord 需要包含<dbxrecrd.h>，但你不必自己添加。

4.1.2 ADDENTRY 命令的代码

在 ObjectARX 向导已经自动生成的程序段中添加代码，使其成下面的样子：

```
// - AsdkStep4._MyADDENTRY command (do not rename)
static void AsdkStep4_MyADDENTRY(void)
{
  // Add your code for command AsdkStep4._MyADDENTRY here
        char strID [133] ;
        if ( acedGetString (0, "Enter employee name: ", strID) != RTNORM )
              return ;

        // Get the named object dictionary
        AcDbDictionary *pNOD ;
        if ( acdbHostApplicationServices()->workingDatabase ()->getNamedObjectsDictionary (pNOD,
AcDb::kForRead) != Acad::eOk ) {
              acutPrintf ("\nUnable to open the NOD! Aborting...") ;
              return ;
        }

        // See if our dictionary is already there
        AcDbObjectId idO ;
        AcDbDictionary *pEmployeeDict =NULL ;
        if ( pNOD->getAt ("ASDK_EMPLOYEE_DICTIONARY", idO) == Acad::eKeyNotFound ) {
```

```
            // Create it if not
            if ( pNOD->upgradeOpen () != Acad::eOk ) {
                acutPrintf ("\nCannot open NOD for Write!") ;
                pNOD->close () ;
                return ;
            }
            pEmployeeDict =new AcDbDictionary ;
            // Add it to the NOD
            if ( pNOD->setAt ("ASDK_EMPLOYEE_DICTIONARY", pEmployeeDict, idO) != Acad::eOk ) {
                // We are really unlucky
                acutPrintf ("\nCannot add our dictionary in the AutoCAD NOD!") ;
                // Clean-up memory and abort
                delete pEmployeeDict ;
                pNOD->close () ;
                return ;
            }
        }
        else {
            // Get it for write if it is already there
            AcDbObject *pO ;
            if ( acdbOpenAcDbObject (pO, idO, AcDb::kForWrite) != Acad::eOk ) {
                acutPrintf ("\nCannot open the object for write.") ;
                pNOD->close () ;
                return ;
            }
            // Check if someone has else has created an entry with our name
            // that is not a dictionary. This should never happen as long as
            // I use the registered developer RDS prefix.
            if ( (pEmployeeDict =AcDbDictionary::cast (pO)) == NULL ) {
                acutPrintf ("\nEntry found in the NOD, but it is not a dictionary.") ;
                pO->close () ;
                pNOD->close () ;
                return ;
            }
        }
        pNOD->close () ;
        // Check if a record with this key is already there
        if ( pEmployeeDict->getAt (strID, idO) == Acad::eOk ) {
            acutPrintf ("\nThis employee is already registered.") ;
            pEmployeeDict->close () ;
            return ;
        }
        // Let's add the new record. Append an empty xrecord.
        AcDbXrecord *pEmployeeEntry =new AcDbXrecord ;
```

```
        if ( pEmployeeDict->setAt (strID, pEmployeeEntry, idO) != Acad::eOk ) {
            acutPrintf ("\nFailed to add the new employee in the dictionary.") ;
            delete pEmployeeEntry ;
            pEmployeeDict->close () ;
            return ;
        }
        pEmployeeEntry->close () ;
        pEmployeeDict->close () ;
    }
```

4.1.3 LISTENTRIES 命令的算法

ObjectARX 向导会在 acrxEntryPoint.cpp 中自动创建将在 ADDENTRY 命令执行时调用的 AsdkStep04_LISTENTRIES()函数。在 AsdkStep04_ LISTENTRIES ()函数中实现下列功能。

1. 在当前工作数据库中得到命名对象词典(AcDbDictionary, AcDbDatabase::getNamedObjectsDictionary())。

2.得到"ASDK_EMPLOYEE_DICTIONARY"词典。(AcDbDictionary::getAt())

3.遍历"ASDK_EMPLOYEE_DICTIONARY"，打印词典的关键字(雇员名)

- 创建新的遍历器(AcDbDictionary::newIterator(), AcDbDictionaryIterator)；
- 遍历 "ASDK_EMPLOYEE_DICTIONARY" (AcDbDictionaryIterator::done(), AcDbDictionaryIterator::next())
- 打印词典的关键字(AcDbDictionaryIterator::name())

4.删除遍历器，不要忘记关闭已打开的对象。

4.1.4 LISTENTRIES 命令的代码

在 ObjectARX 向导已经自动生成的程序段中添加代码，使其成下面的样子：

```
// - AsdkStep4._MyLISTENTRIES command (do not rename)
static void AsdkStep4_MyLISTENTRIES(void)
{
    // Add your code for command AsdkStep4._MyLISTENTRIES here
    AcDbDictionary *pNOD ;
    if ( acdbHostApplicationServices()->workingDatabase ()->getNamedObjectsDictionary (pNOD,
AcDb::kForRead) != Acad::eOk ) {
        acutPrintf ("\nUnable to open the NOD! Aborting...") ;
        return ;
    }
    // See if our dictionary is already there
    AcDbObjectId idO ;
    AcDbObject* pO;
    if ( pNOD->getAt ("ASDK_EMPLOYEE_DICTIONARY", idO) != Acad::eOk ) {
        acutPrintf ("\nNo dictionary, no entry to remove...") ;
        pNOD->close () ;
        return ;
```

```
        }
        // Get employee dictionary for read
        if ( acdbOpenAcDbObject (pO, idO, AcDb::kForRead) != Acad::eOk ) {
            acutPrintf ("\nCannot open the object for write.") ;
            pNOD->close () ;
            return ;
        }
        // Check if someone has else has created an entry with our name
        // that is not a dictionary. This should never happen as long as
        // I use the registered developer RDS prefix.
        AcDbDictionary *pEmployeeDict ;
        if ( (pEmployeeDict =AcDbDictionary::cast (pO)) == NULL ) {
            acutPrintf ("\nEntry found in the NOD, but it is not a dictionary.") ;
            pO->close () ;
            pNOD->close () ;
            return ;
        }
        pNOD->close () ;

        AcDbDictionaryIterator *pIter ;
        if ( (pIter =pEmployeeDict->newIterator (AcRx::kDictCollated)) != NULL ) {
            for ( ; !pIter->done () ;pIter->next () ) {
                // Print name
                acutPrintf ("*Employee: %s\n", pIter->name ()) ;
            }
            delete pIter ;
        }

        pEmployeeDict->close () ;
    }
```

4.1.5 REMOVEENTRY 命令的算法

ObjectARX 向导会在 acrxEntryPoint.cpp 中自动创建将在 ADDENTRY 命令执行时调用的 AsdkStep04_ REMOVEENTRY ()函数。在 AsdkStep04_REMOVEENTRY()函数中实现下列功能。

1.提示用户输入要删除的雇员名(acedGetString())。

2. 在当前工作数据库中得到命名对象词典(AcDbDictionary, AcDbDatabase::getNamedObjectsDictionary())。

3. 得到"ASDK_EMPLOYEE_DICTIONARY"词典。(AcDbDictionary::getAt())

4.用输入的雇员名，得到 AcDbXrecord 条目。

5.如果该雇员条目存在，以写方式打开然后删除它。(AcDbObject::erase())

4.1.6 REMOVEENTRY 命令的代码

在 ObjectARX 向导已经自动生成的程序段中添加代码，使其成下面的样子：

```
// - AsdkStep4._MyREMOVEENTRY command (do not rename)
static void AsdkStep4_MyREMOVEENTRY(void)
{
    // Add your code for command AsdkStep4._MyREMOVEENTRY here
    char strID [133] ;
    if ( acedGetString (0, "Enter employee name: ", strID) != RTNORM )
        return ;
    // Get the named object dictionary
    AcDbDictionary *pNOD ;
    if ( acdbHostApplicationServices()->workingDatabase ()->getNamedObjectsDictionary (pNOD,
AcDb::kForRead) != Acad::eOk ) {
        acutPrintf ("\nUnable to open the NOD! Aborting...") ;
        return ;
    }
    // See if our dictionary is already there
    AcDbObjectId idO ;
    AcDbObject* pO;
    if ( pNOD->getAt ("ASDK_EMPLOYEE_DICTIONARY", idO) != Acad::eOk ) {
        acutPrintf ("\nNo dictionary, no entry to remove...") ;
        pNOD->close () ;
        return ;
    }
    // Get employee dictionary for read
    if ( acdbOpenAcDbObject (pO, idO, AcDb::kForRead) != Acad::eOk ) {
        acutPrintf ("\nCannot open the object for write.") ;
        pNOD->close () ;
        return ;
    }
    // Check if someone has else has created an entry with our name
    // that is not a dictionary. This should never happen as long as
    // I use the registered developer RDS prefix.
    AcDbDictionary *pEmployeeDict ;
    if ( (pEmployeeDict =AcDbDictionary::cast (pO)) == NULL ) {
        acutPrintf ("\nEntry found in the NOD, but it is not a dictionary.") ;
        pO->close () ;
        pNOD->close () ;
        return ;
    }
    pNOD->close () ;
    // Check if a record with this key is there
    if ( pEmployeeDict->getAt (strID, idO) != Acad::eOk ) {
        acutPrintf ("\nEntry not found.") ;
```

```
            pEmployeeDict->close () ;
            return ;
        }
        pEmployeeDict->close () ;
        // Get it for write
        if ( acdbOpenAcDbObject (pO, idO, AcDb::kForWrite) != Acad::eOk ) {
            acutPrintf ("\nEntry cannot be opened for write.") ;
            return ;
        }
        // And erase it
        pO->erase () ;
        pO->close () ;
    }
```

4.2 测试 ADDENTRY，LISTENTRIES 和 REMOVEENTRY 命令

在一个新图中分别执行 ADDENTRY，LISTENTRIES 和 REMOVEENTRY 命令。可以在文本窗口查看检测结果。

```
命令: myADDENTRY
Enter employee name: jj1

命令: MyADDENTRY
Enter employee name: jj2

命令: myLISTENTRIES
*Employee: jj1
*Employee: jj2

命令: myREMOVEENTRY
Enter employee name: jj2

命令: MyLISTENTRIES
*Employee: jj1
```

实例 5. 定制对象_扩展词典

本节我们将关注从 AcDbObject 派生的类。我们的类叫 AsdkEmployeeDetails，被保存到 EMPLOYEE AcDbBlockReference 扩展词典中。AsdkEmployeeDetail 对象将包括雇员的信息。命名 AcDbBlockReference 实体的每一个实例是 AcDbBlockTableRecord 定义的引用。这里，我们将使用前面用过的“EMPLOYEE”块的引用。定制 AsdkEmployeeDetails 类存放雇员的 ID、卧室号、名和姓。下面给出了对象之间的关系：

```
Employee (class: AcDbBlockReference)
        |_ Extension Dictionary (class:AcDbDictionary)
                |_ "ASDK_EMPLOYEE_DICTIONARY" class:AcDbDictionary)
                        |_ "DETAILS" (class:AsdkEmployeeDetails)
```

另外，我们将讨论定制对象和扩展词典，将从对象的一般术语 DB/UI separation 中引入用户界面分离的概念。本节我们将建立两个应用，一个 ObjectDBX 应用说明我们应用的 DB 部分，另一个 ObjectARX 应用说明我们应用的 UI 部分。

本节目标

在本练习中，我们将创建“ADDDETAIL”，″LISTDETAILS″和“REMOVEDETAIL”三个 AutoCAD 命令。

在运行“ADDDETAIL”命令时，该命令将：

- 提示用户选择一个 EMPLOYEE 块引用。
- 如果选择成功，将提示用户输入 EMPLOYEE 的详细信息。
- 得到该实体的扩展词典。
- 检验 EMPLOYEE 块引用的基准″ASDK_EMPLOYEE_DICTIONARY″词典。
- 如果该 EMPLOYEE 块引用不包含 AsdkEmployeeDetails 对象，将创建一个新的 AsdkEmployeeDetails 对象，填入适当的信息，把 AsdkEmployeeDetails 对象加到 EMPLOYEE 块引用宽展词典的基准词典中。

在运行“LISTDETAILS”命令时，该命令将：

- 提示用户选择一个 EMPLOYEE 块引用。
- 如果选择成功，返回 EMPLOYEE 块引用的扩展词典。
- 如果该 EMPLOYEE 块引用有扩展词典，返回 ASDK_EMPLOYEE_DICTIONARY 词典。
- 如果该 EMPLOYEE 块引用有″ASDK_EMPLOYEE_DICTIONARY″词典，返回"DETAILS" AsdkEmployeeDetails 对象。
- 如果该 AsdkEmployeeDetails 对象存在，返回数据并提交给用户。

在运行“REMOVEDETAIL”命令时，该命令将：

- 提示用户选择一个 EMPLOYEE 块引用。
- 如果选择成功，返回 EMPLOYEE 块引用的扩展词典。
- 如果该 EMPLOYEE 块引用有扩展词典，返回 ASDK_EMPLOYEE_DICTIONARY 词典。
- 如果该 EMPLOYEE 块引用有″ASDK_EMPLOYEE_DICTIONARY″词典，返回"DETAILS" AsdkEmployeeDetails 对象。

- 如果该 AsdkEmployeeDetails 对象存在，则删除它。如果没有其他的条目引用"ASDK_EMPLOYEE_DICTIONARY"词典，该词典也被删除。

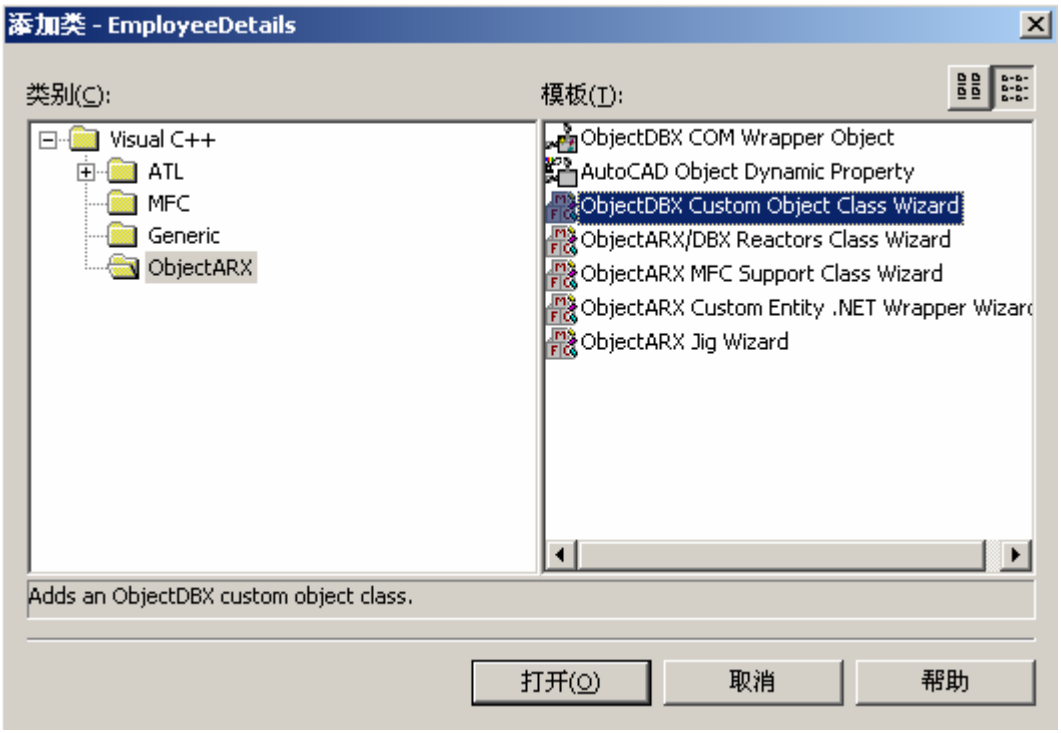
5.1 创建一个 ObjectDBX (*.dbx)模式的子项目

从 Visual C++ .NET 的菜单，选择[文件] >[新建] >[项目…]，弹出“新建项目”对话框。选择"ObjectARX/DBX/OMF Project"模版。键入项目名"Step05"和路径"D:\"，创建 Step05 新项目。

在“解决方案资源管理器”中右键“解决方案 Step05”节点，在快捷菜单中选择[添加]>[新建项目…]，弹出“新建项目”对话框。选择"ObjectARX/DBX/OMF Project"模版。键入项目名"EmployeeDetails"和路径"D:\Step05"，按[确定]弹出“ObjectARX/DBX/OMF 应用向导”对话框。在“Application Type”属性页中选择“ObjectDBX”后，按[Finish]完成 EmployeeDetails 子项目的创建。

5.2 创建 EmployeeDetails 类

5.2.1 右击资源管理器窗口中 EmployeeDetails 项目节点，选择[添加]>[添加类...]，弹出“添加类”对话框。选择"ObjectDBX Custom Object Class Wizard"模版，按[打开]。



5.2.2 在"ObjectDBX Custom Object Class Wizard"对话框中，选择 Names 属性页，创建名为"AsdkEmployeeDetails"的类。注意，前缀“Asdk”已出现在文本框中，我们只需键入 EmployeeDetails。选择基类为"AcDbObject"。

5.2.3 在"ObjectDBX Custom Object Class Wizard"对话框中，选择 Protocols 属性页，选中 DwgProtocol 和 DxfProtocol。按[Finish]向导将从 AcDbObject 类派生 AsdkEmployeeDetails 类。

5.2.4 打开 AsdkEmployeeDetails.h，定义一个常数字符串，通过 acrxDynamicLinker 的 acrxRegisterService()方法，用于注册 ObjectDBX 组件：

```
#define ASDKEMPLOYEEDETAILS_DBXSERVICE "ASDKEMPLOYEEDETAILS_DBXSERVICE"
```

5.3 给 AsdkEmployeeDetails 类添加成员变量和成员函数

观察 AsdkEmployeeDetails.h 文件中 AsdkEmployeeDetails 类的声明，发现向导已经添加了下面一些成员函数（因为在"ObjectDBX Custom Object Class Wizard"中，我们选择了 dwg/dxf 协议）。

```
//----- AcDbObject protocols
//- Dwg Filing protocol
    virtual Acad::ErrorStatus dwgOutFields (AcDbDwgFiler *pFiler) const ;
    virtual Acad::ErrorStatus dwgInFields (AcDbDwgFiler *pFiler) ;
//- Dxf Filing protocol
    virtual Acad::ErrorStatus dxfOutFields (AcDbDxfFiler *pFiler) const ;
    virtual Acad::ErrorStatus dxfInFields (AcDbDxfFiler *pFiler) ;
```

使用这些函数使得能从存储器读写我们定制的类。

5.3.1 添加成员变量

我们给定制的类加入下面表中的私有成员变量：

给 AsdkEmployeeDetails 对象添加的成员变量

Variable Name	Data Type	DXF Code
m_lastName	char*	300
m_firstName	char*	301
m_cube	Adesk::Int32	90
m_ID	Adeak::Int32	91

在 AsdkEmployeeDetails.h 文件，在 AsdkEmployeeDetails 类的声明添加下列代码：

```
private:
    Adesk::Int32 m_ID;
    Adesk::Int32 m_cube;
    char* m_firstName;
    char* m_lastName;
```

在添加类的成员变量时，可以用向导来完成：进入“类视图”窗口，右键 AsdkAsdkEmployeeDetails 类节点，在右键菜单中选择[添加][添加变量…]，弹出“ADD Member Variable Wizard”对话框，在对话框中进行操作即可。

5.3.2 修改构造函数

在 AsdkEmployeeDetails.cpp 文件中，修改 AsdkEmployeeDetails 类的构造函数，使其成为：

```
//-----constructor
AsdkEmployeeDetails::AsdkEmployeeDetails():AcDbObject()
{
```

```
    m_firstName = NULL;
    m_lastName = NULL;
}
```

5.3.3 添加成员函数

我们还必须增加对这些成员变量进行访问/修改的成员函数。在 AsdkEmployeeDetails.h 文件，在 AsdkEmployeeDetails 类的声明中添加下列代码：

```
    //- access/modify functions
    Acad::ErrorStatus setID(const Adesk::Int32 ID);
    Acad::ErrorStatus iD(Adesk::Int32& ID);
    Acad::ErrorStatus setCube(const Adesk::Int32 cube);
    Acad::ErrorStatus cube(Adesk::Int32& cube);
    Acad::ErrorStatus setFirstName(const char* firstName);
    Acad::ErrorStatus firstName(char*& firstName);
    Acad::ErrorStatus setLastName(const char* lastName);
    Acad::ErrorStatus lastName(char*& lastName);
```

在添加类的成员函数时，可以用向导来完成：进入“类视图”窗口，右键 AsdkAsdkEmployeeDetails 类节点，在右键菜单中选择[添加][添加函数…]，弹出“添加成员函数向导”对话框，在对话框中进行操作即可。

5.3.4 完善成员函数

下列代码说明在 AsdkEmployeeDetails.cpp 函数中如何实现上面提到的 firstName()和 setFirstName()函数。

firstName()函数返回存放在 AsdkEmployeeDetails 对象中的 first name：

```
Acad::ErrorStatus AsdkEmployeeDetails::firstName(char *& firstName)
{
    assertReadEnabled();
    firstName = strdup(m_firstName);
    return Acad::eOk;
}
```

setFirstName()函数向 AsdkEmployeeDetails 对象中设置 first name：

```
Acad::ErrorStatus AsdkEmployeeDetails::setFirstName(const char* firstName)
{
    assertWriteEnabled();
    if (m_firstName)
    free(m_firstName);
    m_firstName = strdup(firstName);
    return Acad::eOk;
```

```
}
```

注意，在 firstName()函数中调用的 assertReadEnabled()函数。必须在从对象中读数据前调用这个函数。assertReadEnabled()函数检查对象是否已经以读方式打开，如果还没有打开，该函数将抛出异常，使用户可以保存图形。

类似地，在准备修改对象的函数也应该调用 assertReadEnabled()函数，检查对象是否已经以写方式打开，如果还没有打开，该函数将抛出异常，使用户可以保存图形。

实现其他访问/修改函数的方法是类似的。可**参见 Step05** 项目的代码。

5.4 初始化 AsdkEmployeeDetails 类

打开 EmployeeDetails 项目的 acrxEntryPoint.cpp 文件，用 CEmployeeDetailsApp 类的 On_kInitAppMsg()方法把我们的 DBX 注册为服务。当 DBX 组件卸载时，应该去除我们的服务名。这可以用 On_kInitAppMsg()方法实现。当然，在 acrxEntryPoint.cpp 文件中，应该包括 AsdkEmployeeDetails.h 文件。

```
virtual AcRx::AppRetCode On_kInitAppMsg (void *pkt) {
    AcRx::AppRetCode retCode =AcRxDbxApp::On_kInitAppMsg (pkt) ;
    // TODO: Add your initialization code here
    acrxRegisterService(ASDKEMPLOYEEDETAILS_DBXSERVICE);
    return (retCode) ;
}
virtual AcRx::AppRetCode On_kUnloadAppMsg (void *pkt) {
    AcRx::AppRetCode retCode =AcRxDbxApp::On_kUnloadAppMsg (pkt) ;
    // TODO: Add your code here
    delete acrxServiceDictionary->remove(ASDKEMPLOYEEDETAILS_DBXSERVICE);
    return (retCode) ;
}
```

5.4.1 在加载 ObjectDBX 组件时注册该类

打开 EmployeeDetails 项目的 AsdkEmployeeDetails.h 文件，会看到下列代码：

```
#ifdef EMPLOYEEDETAILS_MODULE
ACDB_REGISTER_OBJECT_ENTRY_AUTO(AsdkEmployeeDetails)
#endif
```

上面的宏将在 ObjectDBX 组件加载和卸载时，负责 AsdkEmployeeDetails 类的注册和删除注册。这将确保在 ObjectDBX 组件被加载时，把我们定制的对象被注册到 ObjectARX 运行类树上。

EmployeeDetails 类可以用下面的代码进行注册。当然，上面的宏为我们代劳了，我们不必使用这些代码。

```
AsdkEmployeeDetails::rxInit();
acrxBuildClassHierarchy();
```

我们可以用下面的代码，从 ObjectARX 运行类树上去除 EmployeeDetails 类。当 ObjectDBX 组件被卸载时，上面的宏仍然负责从 ObjectARX 运行类树上去除 EmployeeDetails 类。

```
deleteAcRxClass(AsdkEmployeeDetails::desc());
```

5.4.2 编译 ObjectDBX 组件

在解决方案资源管理器窗口中右击 EmployeeDetails 项目，选择“Build”，编译该 ObjectDBX 项目。

5.5 实现 ADDDETAIL，LISTDETAILS 和 REMOVEDETAIL 命令

在解决方案资源管理器窗口中选中 Step05 项目。用 ObjectARX 向导创建 ADDDETAIL，LISTDETAILS 和 REMOVEDETAIL 三个命令。

为了方便测试，同时创建在 3.3 节时介绍的 Create 和 setLayer 命令。

5.5.1 ADDDETAIL 命令的算法

ObjectARX 向导会在 acrxEntryPoint.cpp 中自动创建将在 ADDENTRY 命令执行时调用的 AsdkStep05_ADDDETAIL()函数。在 AsdkStep05_ADDDETAIL()函数中实现下列功能。

1.令用户选择一个块引用(acedEntSel())。

- 把 ads_name 转换为 AcDbObjectId(acdbGetObjectId())
- 打开该对象(acdbOpenAcDbObject())
- 用 isKindOf()函数检查用户选择的块引用(AcDbBlockReference::desc())
- 如果不是块引用，返回。

2.输入雇员的 ID，Cube，first name 和 last name (acedGetInt(), acedGetString())

3.获取 EMPLOYEE 块引用的扩展词典(AcDbObject::extensionDictionary())。

4. 如果 EMPLOYEE 块引用没有扩展词典就创建一个(AcDbObject::createExtensionDictionary())。

5.打开扩展字典。如果该扩展字典先前被删除了，则打开这个已删除的扩展字典以便我们重新使用它。

6. 如果扩展字典已被删除(AcDbObject::isErased ())了，则撤消删除(AcDbObject::erase(Adesk::kFalse))

7.找回"ASDK_EMPLOYEE_DICTIONARY" (AcDbDictionary::getAt())。

如果 "ASDK_EMLOYEE_DICTIONARY" AcDbDictionary 不存在，创建一个"ASDK_EMLOYEE_DICTIONARY" AcDbDictionary，加到 EMPLOYEE 块引用扩展词典中。

8. 检查在"ASDK_EMPLOYEE_DICTIONARY" (我们用关键字"DETAILS"存储 AsdkEmployeeDetails 对象)

如果 AsdkEmployeeDetails 对象不存在，创建一个 AsdkEmployeeDetails 对象并设置数据。

用"DETAILS"作为关键字，在"ASDK_EMPLOYEE_DICTIONARY"中存储 AsdkEmployeeDetails 对象

5.5.2 ADDDETAIL 命令的代码

在 ObjectARX 向导已经自动生成的程序段中添加代码，使其成下面的样子：

```
// ----- AsdkStep05._ADDDETAIL command (do not rename)
static void AsdkStep05_ADDDETAIL(void)
{
	// Add your code for command AsdkStep05._ADDDETAIL here
	// Prompt the user for the employee details
	ads_name ename ;
	ads_point pt ;
	// Get the data from the user
	if ( acedEntSel ("Select employee: ", ename, pt) != RTNORM )
		return ;
	// Do a quick check
	// a more comprehensive check could include
	// whether we already have the detail object on this candidate
	AcDbObjectId idO ;
	if ( acdbGetObjectId (idO, ename) != Acad::eOk )
		return ;
	AcDbObject *pO ;
	if ( acdbOpenAcDbObject (pO, idO, AcDb::kForWrite) != Acad::eOk )
		return ;
	if ( !pO->isKindOf (AcDbBlockReference::desc ()) ) {
		acutPrintf ("\nThis is not a block reference.") ;
		pO->close () ;
		return ;
	}
	// Get user input
	int id, cubeNumber ;
	char strFirstName [133] ;
	char strLastName [133] ;
	if (	acedGetInt ("Enter employee ID: ", &id) != RTNORM
		|| acedGetInt ("Enter cube number: ", &cubeNumber) != RTNORM
		|| acedGetString (0, "Enter employee first name: ", strFirstName) != RTNORM
		|| acedGetString (0, "Enter employee last name: ", strLastName) != RTNORM
	) {
		pO->close () ;
		return ;
	}
	// Get the extension dictionary
	if ( (idO =pO->extensionDictionary ()) == AcDbObjectId::kNull ) {
		if ( pO->createExtensionDictionary () != Acad::eOk ) {
			pO->close () ;
			acutPrintf ("\nFailed to create ext. dictionary.") ;
			return ;
		}
		idO =pO->extensionDictionary () ;
```

```
        }
        // We do not need the block reference object anymore.
        pO->close () ;
        // Make sure you open erased extension dictionaries
        // you may need to unerase them
        AcDbDictionary *pExtDict ;
        if ( acdbOpenAcDbObject ((AcDbObject *&)pExtDict, idO, AcDb::kForWrite, Adesk::kTrue) !=
Acad::eOk ) {
            acutPrintf ("\nFailed to open ext. dictionary.") ;
            return ;
        }
        // Unerase the ext. dictionary if it was erased
        if ( pExtDict->isErased () )
            pExtDict->erase (Adesk::kFalse) ;
        // See if our dictionary is already there
        AcDbDictionary *pEmployeeDict ;
        if ( pExtDict->getAt ("ASDK_EMPLOYEE_DICTIONARY", idO) == Acad::eKeyNotFound ) {
            // Create it if not
            pEmployeeDict =new AcDbDictionary ;
            Acad::ErrorStatus es ;
            if ( (es =pExtDict->setAt ("ASDK_EMPLOYEE_DICTIONARY", pEmployeeDict, idO)) !=
Acad::eOk ) {
                pExtDict->close () ;
                acutPrintf ("\nFailed to create the 'Employee' dictionary.") ;
                return ;
            }
        } else {
            // Open our dictionary for write if it is already there
            if ( acdbOpenAcDbObject (pO, idO, AcDb::kForWrite) != Acad::eOk ) {
                pExtDict->close () ;
                acutPrintf ("\nFailed to open the 'Employee' dictionary.") ;
                return ;
            }
            // Check if someone has else has created an entry with our name
            // that is not a dictionary.
            if ( (pEmployeeDict =AcDbDictionary::cast (pO)) == NULL ) {
                pO->close () ;
                pExtDict->close () ;
                acutPrintf ("\nThe entry is not a dictionary") ;
                return ;
            }
        }
        // We do not need the ext. dictionary object anymore
        pExtDict->close () ;
```

```
        // Check if a record with this key is already there
        if ( pEmployeeDict->getAt ("DETAILS", idO) == Acad::eOk ) {
            pEmployeeDict->close () ;
            acutPrintf ("\nDetails already assign to that 'Employee' object.") ;
            return ;
        }
        // Create an EmployeeDetails object and set its fields
        AsdkEmployeeDetails *pEmployeeDetails =new AsdkEmployeeDetails ;
        pEmployeeDetails->setID (id) ;
        pEmployeeDetails->setCube (cubeNumber) ;
        pEmployeeDetails->setFirstName (strFirstName) ;
        pEmployeeDetails->setLastName (strLastName) ;
        // Add it to the dictionary
        if ( pEmployeeDict->setAt ("DETAILS", pEmployeeDetails, idO) != Acad::eOk ) {
            delete pEmployeeDetails ;
            acutPrintf ("\nFailed to add details to that object.") ;
            pEmployeeDict->close () ;
            return ;
        }
        // Done
        acutPrintf ("\nDetails successfully added!") ;
        pEmployeeDict->close () ;
        pEmployeeDetails->close () ;
    }
```

5.5.3 LISTDETAILS 命令的算法

在 AsdkStep05LISTDETAILS ()函数中实现下列功能。

1.令用户选择一个块引用(acedEntSel())。

- 把 ads_name 转换为 AcDbObjectId(acdbGetObjectId())。
- 打开该对象(acdbOpenAcDbObject())。
- 用 isKindOf()函数检查用户选择的块引用(AcDbBlockReference::desc())。
- 如果不是块引用，返回。

2.返回该块引用的扩展词典(AcDbObject::extensionDictionary())。如果不存在则返回。

3.返回"ASDK_EMPLOYEE_DICTIONARY" (AcDbDictionary::getAt())。如果不存在则返回。

4.从"ASDK_EMPLOYEE_DICTIONARY"返回关键字是"DETAILS"的 AsdkEmployeeDetails 对象。

5. 返回 AsdkEmployeeDetails 对象的数据并打印出来。

5.5.4 LISTDETAILS 命令的代码

在 ObjectARX 向导已经自动生成的程序段中添加代码，使其成下面的样子：

```
// ----- AsdkStep05._LISTDETAILS command (do not rename)
```

```
static void AsdkStep05_LISTDETAILS(void)
{
	// Add your code for command AsdkStep05._LISTDETAILS here
	ads_name ename ;
	ads_point pt ;
	// Get the data from the user
	if ( acedEntSel ("Select employee: ", ename, pt) != RTNORM )
		return ;
	// Do a quick check
	// a more comprehensive check could include
	// whether we already have the detail object on this candidate
	AcDbObjectId idO ;
	if ( acdbGetObjectId (idO, ename) != Acad::eOk )
		return ;
	AcDbObject *pO ;
	if ( acdbOpenAcDbObject (pO, idO, AcDb::kForRead) != Acad::eOk )
		return ;
	if ( !pO->isKindOf (AcDbBlockReference::desc ()) ) {
		acutPrintf ("\nThis is not a block reference.") ;
		pO->close () ;
		return ;
	}
	// Get the Ext. Dictionary
	if ( (idO =pO->extensionDictionary ()) == AcDbObjectId::kNull ) {
		// Nothing to do
		pO->close () ;
		return ;
	}
	// We do not need the block reference object anymore.
	pO->close () ;
	// If erased, nothing to do
	AcDbDictionary *pExtDict ;
	if ( acdbOpenAcDbObject ((AcDbObject *&)pExtDict, idO, AcDb::kForRead, Adesk::kFalse) !=
Acad::eOk ) {
		acutPrintf ("\nFailed to open ext. dictionary.") ;
		return ;
	}
	// See if our dictionary is already there
	AcDbDictionary *pEmployeeDict ;
	if ( pExtDict->getAt ("ASDK_EMPLOYEE_DICTIONARY", idO) == Acad::eKeyNotFound ) {
		// Nothing to do if not
		pExtDict->close () ;
		return ;
	} else {
```

```
        // Open dictionary for write if it is already there
        if ( acdbOpenAcDbObject (pO, idO, AcDb::kForRead) != Acad::eOk ) {
            pExtDict->close () ;
            acutPrintf ("\nFailed to open the 'Employee' dictionary.") ;
            return ;
        }
        // Check if someone has else has created an entry with our name
        // that is not a dictionary.
        if ( (pEmployeeDict =AcDbDictionary::cast (pO)) == NULL ) {
            pO->close () ;
            pExtDict->close () ;
            acutPrintf ("\nThe entry is not a dictionary") ;
            return ;
        }
    }
    // Check if a record with this key is already there
    if ( pEmployeeDict->getAt ("DETAILS", idO) != Acad::eOk ) {
        // Nothing to do
        pEmployeeDict->close () ;
        pExtDict->close () ;
        return ;
    }
    // Open the object for write
    if ( acdbOpenAcDbObject (pO, idO, AcDb::kForRead) != Acad::eOk ) {
        pEmployeeDict->close () ;
        pExtDict->close () ;
        acutPrintf ("\nFailed to open the object detail.") ;
        return ;
    }
    // Check it is a AsdkEmployeeDetails object
    AsdkEmployeeDetails *pEmployeeDetails =AsdkEmployeeDetails::cast (pO) ;
    if ( pEmployeeDetails == NULL ) {
        acutPrintf ("\nNo details found!.") ;
        pO->close () ;
        pEmployeeDict->close () ;
        pExtDict->close () ;
        return ;
    }
    // And display details
    Adesk::Int32 i ;
    pEmployeeDetails->iD (i) ;
    acutPrintf ("*Employee's ID: %d\n", i) ;
    pEmployeeDetails->cube (i) ;
    acutPrintf ("*Employee's cube number: %d\n", i) ;
```

```
        char *st =NULL ;
        pEmployeeDetails->firstName (st) ;
        acutPrintf ("*Employee's first name: %s\n", st) ;
        delete [] st ;
        pEmployeeDetails->lastName (st) ;
        acutPrintf ("*Employee's last name: %s\n", st) ;
        delete [] st ;

        pO->close () ;
        pEmployeeDict->close () ;
        pExtDict->close () ;
    }
```

5.5.5 REMOVEDETAIL 命令的算法

在 AsdkStep05LISTDETAILS ()函数中实现下列功能。

1.令用户选择一个块引用(acedEntSel())。

- 把 ads_name 转换为 AcDbObjectId(acdbGetObjectId())。
- 打开该对象(acdbOpenAcDbObject())。
- 用 isKindOf()函数检查用户选择的块引用(AcDbBlockReference::desc())。
- 如果不是块引用，返回。

2.返回该块引用的扩展词典(AcDbObject::extensionDictionary())。如果不存在则返回。

3.返回"ASDK_EMPLOYEE_DICTIONARY" (AcDbDictionary::getAt())。如果不存在则返回。

4.从"ASDK_EMPLOYEE_DICTIONARY"返回关键字是"DETAILS"的 AsdkEmployeeDetails 对象。

5. 删除 AsdkEmployeeDetails 对象(AcDbObject::erase())。

6. 如果当前词典中已没有别的条目，则删除该词典。

5.5.6 REMOVEDETAIL 命令的代码

在 ObjectARX 向导已经自动生成的程序段中添加代码，使其成下面的样子：

```
// ----- AsdkStep05._REMOVEDETAIL command (do not rename)
static void AsdkStep05_REMOVEDETAIL(void)
{
        ads_name ename ;
        ads_point pt ;
        // Get the data from the user
        if ( acedEntSel ("Select employee: ", ename, pt) != RTNORM )
                return ;
        // Do a quick check
        // a more comprehensive check could include
        // whether we already have the detail object on this candidate
        AcDbObjectId idO ;
        if ( acdbGetObjectId (idO, ename) != Acad::eOk )
```

```
            return ;
        AcDbObject *pO ;
        if ( acdbOpenAcDbObject (pO, idO, AcDb::kForRead) != Acad::eOk )
            return ;
        if ( !pO->isKindOf (AcDbBlockReference::desc ()) ) {
            acutPrintf ("\nThis is not a block reference.") ;
            pO->close () ;
            return ;
        }
        // Get the Ext. Dictionary
        if ( (idO =pO->extensionDictionary ()) == AcDbObjectId::kNull ) {
            // Nothing to do
            pO->close () ;
            return ;
        }
        // We do not need the block reference object anymore.
        pO->close () ;
        // If erased, nothing to do
        AcDbDictionary *pExtDict ;
        if ( acdbOpenAcDbObject ((AcDbObject *&)pExtDict, idO, AcDb::kForWrite, Adesk::kFalse) !=
Acad::eOk ) {
            acutPrintf ("\nFailed to open ext. dictionary.") ;
            return ;
        }
        // See if our dictionary is already there
        AcDbDictionary *pEmployeeDict ;
        if ( pExtDict->getAt ("ASDK_EMPLOYEE_DICTIONARY", idO) == Acad::eKeyNotFound ) {
            // Nothing to do if not
            pExtDict->close () ;
            return ;
        } else {
            // Open the dictionary for write if it is already there
            if ( acdbOpenAcDbObject (pO, idO, AcDb::kForWrite) != Acad::eOk ) {
                pExtDict->close () ;
                acutPrintf ("\nFailed to open the 'Employee' dictionary.") ;
                return ;
            }
            // Check if someone has else has created an entry with our name
            // that is not a dictionary.
            if ( (pEmployeeDict =AcDbDictionary::cast (pO)) == NULL ) {
                pO->close () ;
                pExtDict->close () ;
                acutPrintf ("\nThe entry is not a dictionary") ;
                return ;
```

```
            }
        }
        // Check if a record with this key is already there
        if ( pEmployeeDict->getAt ("DETAILS", idO) != Acad::eOk ) {
            pEmployeeDict->close () ;
            pExtDict->close () ;
            acutPrintf ("\nNo details assigned to that 'Employee' object.") ;
            return ;
        }
        // Open the object for write
        if ( acdbOpenAcDbObject (pO, idO, AcDb::kForWrite) != Acad::eOk ) {
            pEmployeeDict->close () ;
            pExtDict->close () ;
            acutPrintf ("\nFailed to open the object detail.") ;
            return ;
        }
        // And erase it
        pO->erase () ;
        pO->close () ;
        // Erase dictionary if it has no more entries
        if ( pEmployeeDict->numEntries () == 0 )
            pEmployeeDict->erase () ;
        pEmployeeDict->close () ;
        // Erase ext. dictionary if it has no more entries
        if ( pExtDict->numEntries () == 0 )
            pExtDict->erase () ;
        pExtDict->close () ;
    }
```

5.6 加载 AsdkEmployeeDetails.dbx 的 ObjectDBX 组件

在编译 AsdkEmployeeDetails.dbx 过程中，需要编译加载 ObjectDBX 组件的 ObjectARX 应用。

5.6.1 创建 2 个 def 文件

与 1.5.1 节的介绍类似，为 Step05 项目创建一个模块定义文件 Step05.def，同时为 EmployeeDetails 子项目也创建一个模块定义文件 EmployeeDetails.def。两个 def 文件的内容相同，如下所示。

```
EXPORTS
        acrxEntryPoint              PRIVATE
        acrxGetApiVersion           PRIVATE
```

注意，要把向导自动生成的第一行 LIBRARY Step05/ LIBRARY EmployeeDetails 删掉，否则将不能加载 AsdkStep05.Arx。

5.6.2 把 AsdkEmployeeDetails 对象加入命令模块

打开 Step05 项目的属性页窗口。

在链接器的输入项的附加依赖项设置中加入 AsdkEmployeeDetails.lib 库。

在链接器的输入项的模块定义文件项设置中加入 Step05.def 定义文件。（图 5.6）

打开 kEmployeeDetails 子项目的属性页窗口。

在链接器的输入项的模块定义文件项设置中加入 kEmployeeDetails.def 定义文件。

对 Step05 和 kEmployeeDetails 两者，在配置时都要对 Debug 和 Release 状态分别进行设置。注意，库的相对路径也是分别对应 Debug 文件夹和 Release 文件夹。

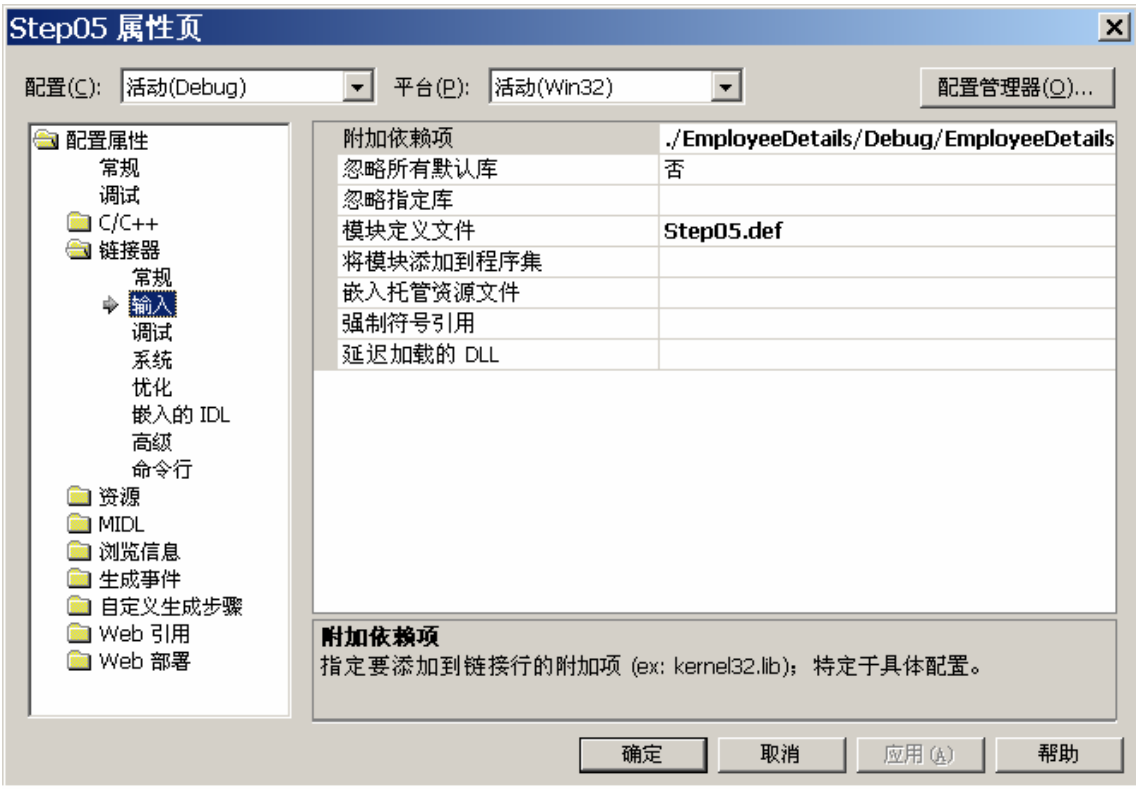


图5.6 项目属性页中链接器的设置

5.7 测试 AsdkEmployeeDetails.dbx 和 AsdkStep05.arx 应用

先后编译生成 AdskEmployeeDetails.dbx 和 AsdkStep05.arx。为了测试本应用，在 AutoCAD 中用 APPLOAD 命令**先后载入** AsdkEmployeeDetails.dbx 组件和 AsdkStep05.arx 应用。

（1）首先运行 Create 命令，创建 EMPLOYEE 块；

（2）用 Acad 的 Insert 命令，插入几个 EMPLOYEE 块引用；

（3）然后运行 ADDDETAIL 命令，给每个雇员（EMPLOYEE 块）加入 ID，Cube 等数据；

（4）运行 LISTDETAILS 查看雇员的数据；

（5）运行 REMOVEDETAIL 命令，删除雇员的数据；

（6）运行 LISTDETAILS 查看删除后的结果。

实例 6.定制实体

本节我们将创建一个定制的实体。定制的实体是一个从 AcDbEllipse 派生的 AsdkEmployee 实体。在这里，我们将从派生得到 AcDbEllipse 对象的所有性质。和以前一样，定制的 AsdkEmployee 类将保存一个雇员的 ID、卧室号、名和姓。但是，这些信息将保存在实体本身的内部，而不是像以前那样保存在 NOD 或扩展词典中。本应用和上一个应用一样，也是一个使用 DB/UI 分离的例子。我们将创建一个 ObjectARX 和 ObjectDBX 应用。

本节目标

在本练习中，我们将只创建一个“CREATEEMPLOYEE”命令。在 CREATEEMPLOYEE 命令执行过程中：

- ◆ 提示用户输入一个雇员的 ID、卧室号、名、姓和地址。
- ◆ 在模型空间放置一个用户定制的带有用户输入的信息的 AsdkEmployee 对象。

在练习的开始，先用 ObjectARX 向导创建 Step06 ObjectARX 项目。

6.1 创建一个 ObjectDBX (*.dbx)模式的子项目

按照 5.1 节的步骤，创建一个名为 Employee 的 ObjectDBX 子项目。

6.2 创建 AsdkEmployee 类

用 ObjectARX 向导创建一个定制实体的步骤与 5.2 节中创建一个定制对象类似。

6.2.1 右击资源管理器窗口中 Employee 项目节点，选择[添加]>[添加类…]，弹出“添加类”对话框。选择"ObjectDBX Custom Object Class Wizard"模版，按[打开]。

6.2.2 在弹出的"ObjectDBX Custom Object Class Wizard"对话框中，创建一个名为"AsdkEmployee"的类。选择"AcDbEllipse"做为基类。

6.2.3 选择"Protocols"属性页，选中"Dwg Protocol"和"Dxf Protocol"。点击[Finish]，向导将创建一个从 AcDbEllipse 类派生的 AsdkEmployee 类。

6.2.4 打开 AsdkEmployee.h，我们要定义一个常数字符串，通过 acrxDynamicLinker 的 acrxRegisterService()方法，用于注册 ObjectDBX 组件。

```
#define ASDKEMPLOYEE_DBXSERVICE "ASDKEMPLOYEE_DBXSERVICE"
```

6.3 给 AsdkEmployee 类添加成员函数

6.3.1 给 AsdkEmployee 类添加成员变量

给 AsdkEmployee 类添加下列成员变量：

变量名	数据类型	DXF 码
m_lastName	char*	300
m_firstName	char*	301
m_cube	Adesk::Int32	90
m_ID	Adeak::Int32	91

1. 在 AsdkEmployee.h 头文件添加 AsdkEmployee 的下列声明：

```
private:
Adesk::Int32 m_ID;
Adesk::Int32 m_cube;
char* m_firstName;
char* m_lastName
```

2．我们要给成员变量添加访问/修改函数。在 AsdkEmployee 的声明中添加下列代码：

```
public:
    Acad::ErrorStatus setID(const Adesk::Int32 ID);
    Acad::ErrorStatus iD(Adesk::Int32& ID);
    Acad::ErrorStatus setCube(const Adesk::Int32 cube);
    Acad::ErrorStatus cube(Adesk::Int32& cube);
    Acad::ErrorStatus setFirstName(const char* firstName);
    Acad::ErrorStatus firstName(char*& firstName);
    Acad::ErrorStatus setLastName(const char* lastName);
   Acad::ErrorStatus lastName(char*& lastName);
```

3.下列代码说明如何实现上列函数。firstName()将返回存储在 AsdkEmployee 对象中的雇员名，setFirstName()将给对象设置雇员名。

```
Acad::ErrorStatus AsdkEmployeeDetails::firstName(char *& firstName)
{
    assertReadEnabled();
    firstName = strdup(m_firstName);
    return Acad::eOk;
}

Acad::ErrorStatus AsdkEmployeeDetails::setFirstName(const char* firstName)
{
    assertWriteEnabled();
    if (m_firstName)
        free(m_firstName);
    m_firstName = strdup(firstName);
    return Acad::eOk;
}
```

4．类似地，实现其余的访问/修改函数。

5．实现 worldDraw()函数。

ObjectARX 向导已经为用户实现了 worldDraw()函数的基本框架。在这个函数中，必须首先调用父类(AcDbEllipse::worldDraw())的基类函数 worldDraw()。然后，以文本方式绘制 AsdkEmployee 实体。(AcGiWorldDraw::geometry(), AcGiGeometry::text())。提示：用 mode->geometry().text ，向 worldDraw()传递一个指针。文本的位置、法线、方向和高度采

用从 AcDbEllipse 派生的 AsdkEmployee 实体的属性。完善后的 worldDraw()函数如下所示：

```
Adesk::Boolean AsdkEmployee::worldDraw (AcGiWorldDraw *mode) {
        assertReadEnabled();
        //----- Draw the AcDbEllipse entity
        AcDbEllipse::worldDraw (mode) ;
        //----- Write the Employee ID and Name
        char buffer [255] ;
        sprintf (buffer, "%d (cube#: %d)", m_ID, m_cube) ;
        mode->geometry ().text (center (), normal (), majorAxis (), minorAxis ().length () / 2, 1.0, 0.0, buffer) ;
        sprintf (buffer, "%s %s", m_firstName, m_lastName) ;
       mode->geometry ().text (center () - minorAxis () / 2, normal (), majorAxis (), minorAxis ().length () / 2, 1.0, 0.0,
buffer) ;
        //----- Returns Adesk::kTrue to not call viewportDraw()
        return (Adesk::kTrue) ;
}
```

6. 如下所示修改 AsdkEmployee 的构造函数：

```
AsdkEmployee::AsdkEmployee() : AcDbEllipse (AcGePoint3d (),AcGeVector3d (0, 0, 1), AcGeVector3d (1, 0, 0),
1)
{
        // TODO: do your initialization.
        m_firstName = m_lastName = NULL ;
}
```

尝试完成上列的其他函数。可参考 ObjectARX 在线帮助和 Step06 项目。

6.3.2 给 AsdkEmployee 类添加初始化代码

下面需要给 AsdkEmployee 添加初始化代码，这与 AsdkEmployeeDetials 类的初始化类似。

打开 acrxEntryPoint.cpp 文件，加上包含 AsdkEmployee.h 的语句。在 CemployeeApp 类的 On_kInitAppMsg()方法中添加下列代码。我们还要从 acrxDynamicLinker 中去掉我们的服务名，可以在 On_kUnloadAppMsg()中实现。

```
virtual AcRx::AppRetCode On_kInitAppMsg (void *pkt) {
        AcRx::AppRetCode retCode =AcRxDbxApp::On_kInitAppMsg (pkt) ;
        // TODO: Add your initialization code here
        acrxRegisterService(ASDKEMPLOYEE_DBXSERVICE);
        return (retCode) ;
}
virtual AcRx::AppRetCode On_kUnloadAppMsg (void *pkt) {
        AcRx::AppRetCode retCode =AcRxDbxApp::On_kUnloadAppMsg (pkt) ;
        // TODO: Add your code here
        delete acrxServiceDictionary->remove(ASDKEMPLOYEE_DBXSERVICE);
```

```
    return (retCode) ;
}
```

6.3.3 编译 ObjectDBX 组件

在类管理器中右键 Employee 项目，选择"Build"，编译该 ObjectDBX 项目。

6.4 实现 CREATEEMPLOYEE 命令

在解决方案资源管理器窗口中选中 Step06 项目。用 ObjectARX 向导创建 CREATEEMPLOYEE 命令。

6.4.1 CREATEEMPLOYEE 命令的实现

- 输入雇员的 ID，Cube，first name，last name (acedGetInt(), acedGetString()) 和雇员的位置(acedGetPoint())。注意：acedGetPoint()返回一个 ads_point 代表用户拾取的点。用全局函数 asDblArray()把 ads_point 直接转换为 AcGePoint3d 点。参见关于 asDblArray()的在线帮助。
- 创建 AsdkEmployee 实体的一个实例。
- 设置 AsdkEmployee 的数据。
- 把 AsdkEmployee 实体加入模型空间记录。

尝试自己完成所有的函数。可参考 ObjectARX 在线帮助和 Step06 项目。

6.4.2 加载 AsdkEmployee.dbx ObjectDBX 组件

加载 ObjectDBX 组件的代码和步骤都是类似的。可参见上面的 5.6.1 节和 5.6.2 节，以便在你的 Step06 ARX 项目中编写类似的代码。

6.5 测试 AsdkEmployee.dbx 和 AsdkStep06.arx 应用

生成 ARX 应用 AsdkStep06.arx。在 AutoCAD 中用 APPLOAD 命令先后载入 AsdkEmployee.dbx 组件和 AsdkStep05.arx 应用。用 CREATEEMPLOYEE 创建几个 EMPLOYEE 实体，保存图形文件。卸载该应用和 dbx 组件。注意到有一个代理实体。重新载入然后 dbx 组件和应用，注意到 EMPLOYEE 再一次成为有效的实体。

实例 7.临时反应器

本节我们将讨论临时反应器。反应器被放在 AutoCAD 环境中，对 AutoCAD 中发生的事件进行监视并做出反应。反应器包括多种类型。在这节，我们将讨论 4 种类型的反应器：

1．数据库反应器 AcDbDatabaseReactor

数据库反应器监视在 AutoCAD 数据库范围内的事件，例如添加或去除实体。

2．AutoCAD 编辑反应器 AcDbEditorReactor

编辑反应器监视在用户编辑图形文件时发生的事件，例如发出命令，或关闭一个当前的活动。

3．对象反应器 AcDbobjectReactor

对象反应器可以监视在指定对象被更改、复制或删除时发生的事件。

4．AutoCAD 文档管理反应器 AcApDocManagerReactor

从 R2000 以来，AutoCAD 是一个多文档环境。文档管理反应器监视在 MDI 模式下图形文档窗口事件，例如打开、关闭、切换图形文档等等。也支持单文档模式的事件。

本节目标

在本练习中，这一次创建的应用不注册什么新的 AutoCAD 命令，而只靠我们在第 3 节开发的 EMPLOYEE 块引用，在 AutoCAD 图形中插入 EMPLOYEE 块引用。不论何时用户试图移动一个 EMPLOYEE 块引用，本程序将重新设置 EMPLOYEE 块引用到它早先的位置。这个功能的实现就需要使用编辑反应器。当某个 AutoCAD 命令修改 EMPLOYEE 块引用的位置时，编辑反应器就会被通知。如果某个编辑命令作用于一个或几个 EMPLOYEE 块应用，这些块的早先和最终的位置将被记录下来。

为达到这个目的，为每个 EMPLOYEE 块引用设置一个对象反应器以监测该块是否被修改操作所打开。在实际操作发生前就可以监测到。当操作发生后，为确定新的位置，我们还要实现编辑反应器的命令结束通知。

最后，我们还要监测 EMPLOYEE 块引用是何时创建的，可以用数据库反应器来达到这一目的。

记住，用户可能打算对一个选择集使用编辑命令，该选择集除了其他实体外可能包括一个或几个 EMPLOYEE 块引用。为了处理这种情况，我们需要为要被修改的 EMPLOYEE 块引用建立对象 ID 数组（AcDbObjectIdArray）。与此类似，我们也需要维护一个指针数组(AGePoint3dArray)来保存 EMPLOYEE 块引用早先位置的指针(AGePoint3dArray)。

为管理好 AcDbObjectIdArray 和 AcGePoint3dArray，我们需要利用文档原理管理文档信息。每个文档管理自己的 AcDbObjectIdArray 和 AcGePoint3dArray 数据集。所以，我们需要一个文档管理器在文档切换时来管理对应得数据集。

做为本练习的起点，下载 Step07 模板项目。模板项目中已经有了供我们测试代码的必要命令。

7.1 准备文档数据

ObjectARX 向导自动创建 CdocData 类（参见 DocData.h 和 DocData.cpp）。CdocData 类可用于实现从属于文档的数据。

在 DocData.h 中给 CdocData 类加入下列成员变量：

数据类型	变量名

bool	m_editCommand
bool	m_doRepositioning
AcDbObjectIdArray	m_changedObjects
AcGePoint3dArray	m_employeePositions

m_editCommand 用于记录是否我们监视的某个命令是活动的。

m_doRepositioning 用于使我们知道是否某个 EMPLOYEE 酷爱引用被要求重新定位。

changedObjects 用于记录那个 EMPLOYEE 块引用被修改了。

employeePositions 用于保存 EMPLOYEE 块引用原来的位置。

在 DocData.cpp 中，CdocData 的构造函数中，我们将初始设置：

```
m_editCommand = false;
m_doRepositioning = false;
```

注意：在创建 Step07 项目时，向导已经用语句“AcApDataManager<CDocData> DocVars ； ”声明了一个全局变量 AsdkDataManager DocVars。

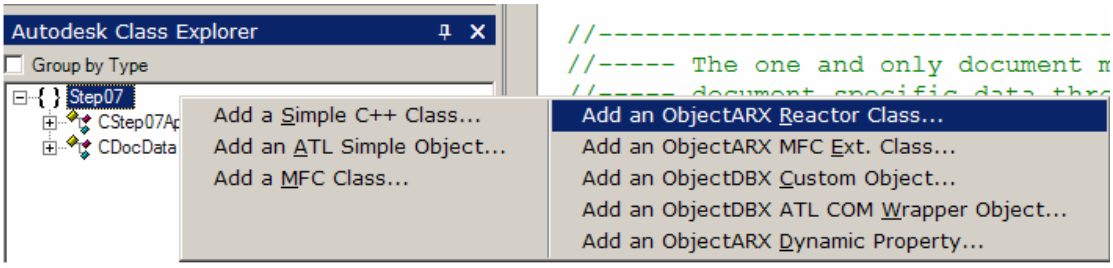
通过 DocVars.docData()达到访问文档数据的目的。

7.2 为项目加入和实现编辑反应器

7.2.1 调用 ObjectARX 反应器向导

可采用两种方法之一调用 ObjectARX 反应器向导：

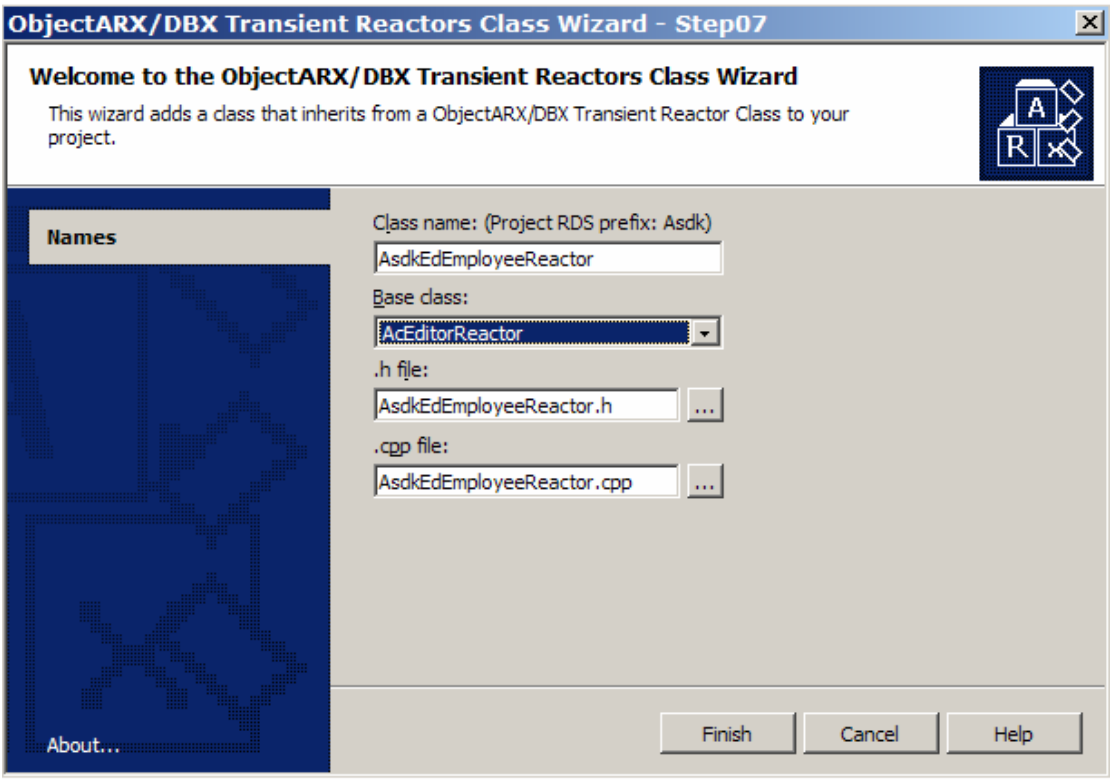
1.在 Autodesk 类管理器窗口中，右键 Step07 项目，选择"Add An ObjectARX Reactor Class"菜单项。



使用ACE加入ObjectARX反应器

2．在解决方案管理器窗口，右键 Step07 项目，选择"Add Class"。弹出添加类对话框，选择"ObjectARX/DBX Reactors Class Wizard"类模板。

在弹出的 ObjectARX/DBX Reactors Class Wizard 对话框中，基类选为 AcEditorReactor，在类名编辑框中键入 AsdkEdEmployeeReactor。

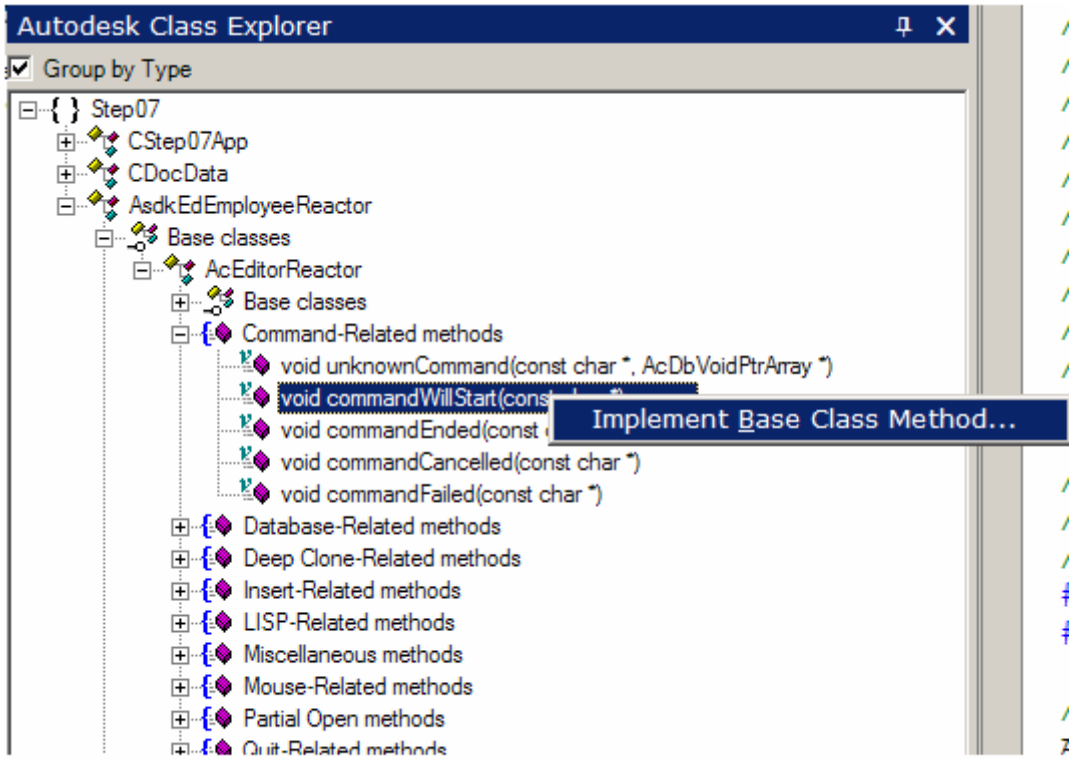


用ObjectARX/DBX临时反应器类向向导加入反应器

7.2.2 实现基类 AcEditorReactor 的虚拟函数

- commandWillStart()
- commandEnded()

可以使用 Autodesk 类管理器实现这些虚拟函数。选中"Group by Type"项，展开基类 AcEditorReactor 的节点。选择"Command-Related methods"节点。右键 commandWillStart()节点，选择"Implement Base Class Method"菜单项。用类似的操作实现 commandEnded 方法。



用ACE在派生类中实现基类的虚拟函数

7.2.3 给 commandWillStart()添加代码

a.检测是否有"MOVE", "ROTATE", "STRETCH", "SCALE" 或 "GRIP_STRETCH"命令之一被启动（使用象 strcmp()那样的字符串比较函数）。

B.如果是被监视的命令之一，设置 m_editCommand = true，m_doRepositioning = false（使用 DocVars.docData().<这儿是适当的变量>）。

c.重新设置所有的对象 ID 和位置信息为 0（AcDbObjectIdArray::setLogicalLength()）。

```
void AsdkEdEmployeeReactor::commandWillStart(const char * cmdStr)
{
    // Check whether one of our monitored commands starts
    if (    strcmp (cmdStr, "MOVE") && strcmp (cmdStr, "ROTATE")
         && strcmp (cmdStr, "SCALE") && strcmp (cmdStr, "STRETCH")
         && strcmp (cmdStr, "GRIP_STRETCH")
    ) {
        return ;
    }
    // Set the global variable...
    DocVars.docData ().m_editCommand =true ;
    DocVars.docData ().m_doRepositioning =false ;
    // ...and delete all stored information
    DocVars.docData ().m_changedObjects.setLogicalLength (0) ;
    DocVars.docData ().m_employeePositions.setLogicalLength (0) ;
```

```
}
```

7.2.4 给 commandEnded ()添加代码

a. 如果被监视的命令不是上面列出的要监视的命令之一，设置 m_editCommand = false，然后返回。

b. 重新设置 m_editCommand 为 false。

c. 设置 m_doRepositioning 为 true，并开始重新定位被移动的对象。

d.对 m_changedObjects (AcDbObjectIdArray::length())中的每一个对象，做如下操作：

(1)打开该实体(acdbOpenObject()).。

(2)如果实际位置与保存的位置不同，重新定位该对象 (AcDbBlockReference::position(), m_employeePositions.at(), AcDbBlockReference::setPosition()) 。

```
void AsdkEdEmployeeReactor::commandEnded(const char * cmdStr)
{
    // Was one of our monitored commands active
    if ( DocVars.docData ().m_editCommand == false )
        return;

    DocVars.docData ().m_editCommand =false ;
    // disable the object reactor
    // (so the object reactor knows that we are changing the object)
    DocVars.docData ().m_doRepositioning =true ;
    // Reset to previous position
    AcDbBlockReference *pInsert ;
    for ( int i =0 ; i < DocVars.docData ().m_changedObjects.length () ; ++i ) {
        if ( acdbOpenObject (pInsert, DocVars.docData ().m_changedObjects.at (i), AcDb::kForWrite) ==
Acad::eOk ) {
            AcGePoint3d newPos =pInsert->position () ;
            AcGePoint3d oldPos =DocVars.docData ().m_employeePositions.at (i) ;
            // Resetting the position to the original one
            if ( newPos != oldPos ) {
                pInsert->setPosition (oldPos) ;
                acutPrintf ("\nEMPLOYEE has been reset to its original location.") ;
            }
            pInsert->close () ;
        }
    }
}
```

7.2.5 创建 AsdkEdEmployeeReactor 反应器实例

在 StdAfx.h 文件中，加入下列代码：

```
#include "AsdkEdEmployeeReactor.h"
extern AsdkEdEmployeeReactor *pEdEmployeeReactor ;
```

a.创建一个全局指针指向你的编辑反应器。把这个反应器放入 acrxEntryPoint.cpp 文件中：

```
AsdkEdEmployeeReactor *pEdEmployeeReactor = NULL;
```

b.当应用被加载时，要创建 AsdkEdEmployeeReactor 编辑反应器的一个实例，在 acrxEntryPoint.cpp 文件，CStep07App 类的 On_kInitAppMsg()方法中写入下列代码：

```
pEdEmployeeReactor = new AsdkEdEmployeeReactor(true);
```

注意：一般情况下，在我们应用的生命期内，当临时反应器工作时，我们利用 addReactor()和 removeReactor()函数在适当的时机设置或移除临时反应器。特别地，倘若构造函数的自变量是 true，由 ObjectARX 向导和 ObjectDBX 向导创建的 AsdkEdEmployeeReactor 类把添加/移除 AsdkEdEmployeeReactor 的职责分别委派给类的构造/析构函数完成。要详细了解 AsdkEdEmployeeReactor.cpp 文件。当加载应用时加入编辑反应器，当卸载应用时移除编辑反应器。在卸载应用时移除编辑反应器是非常重要的，否则会导致 AutoCAD 崩溃。

c.卸载应用时，删除编辑反应器。在 acrxEntryPoint.cpp 文件，CStep07App 类的 On_kUnloadAppMsg ()方法中写入下列代码：

```
delete pEdEmployeeReactor;
```

注意：这里我们创建了一个 AsdkEmployeeReactor (一个 AcDbObjectReactor 对象)，下面将会看到，我们把这个实例附着到 EMPLOYEE 块参考上去。我们也可能创建新的 AsdkEmployeeReactor 实例，把每个实例附着到每个已创建的 EMPLOYEE 块引用上。但你要知道，一个 AcDbObjectReactor 可以附着到多个对象上去。

尝试自己实现所有其他的函数。可参考 ObjectARX 在线帮助和 Step07 项目。

7.3 为项目加入和实现对象反应器

7.3.1 创建对象反应器

利用 ObjectARX 向导创建从 AcDbObjectReactor 派生的对象反应器 AsdkEmployeeReactor。创建对象反应器的步骤和用 ObjectARX 向导创建编辑反应器类似。可参考上节的内容。

7.3.2 实现 openedForModify()函数

用 ObjectARX 向导，实现基类(AcDbObjectReactor)的函数 openedForModify()。检查由 ObjectARX 向导生成的头文件“AsdkEmployeeReactor.h”和 AsdkEmployeeReactor.cpp 文件。

7.3.3 给 openedForModify()添加代码

a.如果当前正在重新定位(m_doRepositioning == true)则返回。(DocVars.docData().<变量>)

b.如果没有被监测的命令被执行(m_editCommand == false)则返回。

c.校验被传递的 AcDbObject 是否为 AcDbBlockReference 实体，如果否就返回。

d.得到 AcDbBlockReference 的名字。因此，必须得到 AcDbBlockReference 块定义的 AcDbBlockTableRecord ， 然 后 查 询 AcDbBlockTableRecord 的 名 字 （ 用 AcDbBlockReference::blockTableRecord ()）。如果不是 EMPLOYEE 块参考，立即返回。

e.保存 openedForModify()对象的位置 m_employeePositions(AcGePoint3dArray::append())

f.保存 openedForModify()对象的 ID m_changedObjects (AcDbObjectIdArray::append())。

```
void AsdkEmployeeReactor::openedForModify(const AcDbObject * dbObj)
{
    if ( DocVars.docData ().m_doRepositioning )
        return ;
    // If none of our monitored commands is active return
    if ( !DocVars.docData ().m_editCommand )
        return ;
   // dbObj should be of type AcDbBlockReference
    // whose name is "EMPLOYEE". This check is not absolutely necessary,
    // since our application is adding the reactor to employee objects
    // only, we know this notificatin can only come from such an object.

    AcDbBlockReference *pInsert =AcDbBlockReference::cast (dbObj) ;
    if ( !pInsert )
        return ;
    // Get the ObjectId of the BlockTableRecord where the reference is defined
    AcDbObjectId blockId =pInsert->blockTableRecord () ;
    AcDbBlockTableRecord *pBlockTableRecord ;
    if ( acdbOpenAcDbObject ((AcDbObject *&)pBlockTableRecord, blockId, AcDb::kForRead) !=
Acad::eOk ) {
        acutPrintf ("\nCannot open block table record!") ;
        return ;
    }
    const char *blockName ;
    pBlockTableRecord->getName (blockName) ;
    pBlockTableRecord->close () ;

    if ( strcmp (blockName, "EMPLOYEE") )
        return ; // Not an employee
   // Store the objectID and the position
    DocVars.docData ().m_changedObjects.append (pInsert->objectId ()) ;
    DocVars.docData ().m_employeePositions.append (pInsert->position ()) ;
}
```

7.3.4 创建 AsdkEmployeeReactor 反应器的]实例

a.创建一个指向对象反应器的全局指针。把反应器放到 acrxEntryPoint.cpp 文件中：

```
AsdkEmployeeReactor *pEmployeeReactor = NULL;
```

b.当应用被加载时创建一个对象反应器实例，在 acrxEntryPoint.cpp 文件，CStep07App 类的 On_kInitAppMsg()方法中写入下列代码：

```
pEmployeeReactor = new AsdkEmployeeReactor();
```

c.如果应用被卸载，删除该对象反应器。在 acrxEntryPoint.cpp 文件，CStep07App 类的 On_kUnloadAppMsg ()方法中写入下列代码：

```
delete pEmployeeReactor;
```

注意：这里我们创建了一个 AsdkEmployeeReactor (一个 AcDbObjectReactor 对象)，下面将会看到，我们把这个实例附着到 EMPLOYEE 块参考上去。我们也可能创建新的 AsdkEmployeeReactor 实例，把每个实例附着到每个已创建的 EMPLOYEE 块引用上。但你要知道，一个 AcDbObjectReactor 可以附着到多个对象上去。

尝试自己实现所有其他的函数。可参考 ObjectARX 在线帮助和 Step07 项目。

7.4 实现应用函数

7.4.1 应用函数 attachEmployeeReactorToAllEmployee()的实现

注意，正如早先提到的，我们用 addReactor()/removeReactor()向对象添加/去除临时反应器。在这里为我们准备把反应器附着到所有 EMPLOYEE 快引用对象上去。记住，可能有多个打开的图形文件中含有 EMPLOYEE 快引用。AsdkDataManager 类管理 CdocData 类，每个文档有一个 CdocData 类。在 CdocData 类的结构函数中，调用我们定义的函数 attachEmployeeReactorToAllEmployee()。在当前 AutoCAD 环境下，每个打开的图形文件、新创建的图形文件和打开已有的图形文件都要调用。设个函数将负责为每个图形文件附着/去除 AsdkEmployeeReactor 对象。

1.向图形文件中已有的所有 EMPLOYEE 引用附着对象反应器。在 docData.cpp 中写出全局函数 Acad::ErrorStatus attachEmployeeReactorToAllEmployee(bool attach)，该函数将：

a.得到块表(AcDbDatabase::getBlockTable())。

b.得到模型空间记录(AcDbBlockTable::getAt(), ACDB_MODEL_SPACE)。

c.得到块表记录遍历器(AcDbBlockTableRecordIterator, AcDbBlockTableRecord::newIterator())。

d.遍历模型空间记录并得到实体(AcDbBlockTableRecordIterator::getEntity())。如果块引用实体不是 EMPLOYEE 对象，立即返回。

e.如果得到的实体是 EMPLOYEE 块引用，附着对象反应器(AcDbObject::addReactor())，如果参数 attach 的值是 false 则去除已经附着的对象反应器(AcDbObject::removeReactor())。

f.最后，删除遍历器，关闭对象。

2.在 CdocData 的结构函数里调用 attachEmployeeReactorToAllEmployee()。

7.4.2 应用函数 detachEmployeeReactorToAllEmployee()的实现

1.如果卸载应用就必须拆掉附着在 EMPLOYEE 块引用实体上的对象反应器。在 docData.cpp 中写出全局函数 void detachAllEmployeeReactors()函数，该函数将：

a.得到遍历器 AcApDocumentIterator (acDocManager->newAcApDocumentIterator())。

acDocManager 是一个全局宏，用于得到管理 AutoCAD 中所有文档的 AcApDocManager。

b. 保存当前文档的指针，使得在遍历以后可以重新设置文档(acDocManager->curDocument())。

c.遍历所有的文档，调用 attachEmployeeReactorToAllEmployee(false)。

d. 遍历以后，把遍历前的当前文档重新设置为当前文档。

2.在应用卸载时，调用 detachAllEmployeeReactors()。

尝试自己实现所有其他的函数。可参考 ObjectARX 在线帮助和 Step07 项目。

7.5 为应用加入和实现数据库反应器

7.5.1 创建数据库反应器

利用 ObjectARX 向导创建从 AcDbDatabaseReactor 派生的数据库反应器 AsdkDbEmployeeReactor。ObjectARX 向导会自动给新类加上 RDS 前缀。

7.5.2 实现 objectAppended()函数

用 ObjectARX 向导，实现基类 AcDbDatabaseReactor 的虚拟函数 objectAppended ()。在这个函数加入下列代码：

a.检查要添加的对象是否是 AcDbBlockReference(cast)。如果否则返回。

b.得到引用的块表记录(AcDbBlockReference::blockTableRecord())。

c. 得到块表记录名，检查其是否是 EMPLOYEE 块引用，如果否则返回。(AcDbSymbolTable::getName())。

d.向添加的块引用附着 AsdkEmployeeReactor 对象反应器(AcDbObject::addReactor())。记住，我们一直在注意数据库是否有 EMPLOYEE 块引用被添加进来。如果有 EMPLOYEE 块引用被添加进来，就向其附着 AsdkEmployeeReactor 对象反应器。

7.5.3 加入指针变量

在 CdocData 中加入指针变量，指向每个文档的 AsdkDbEmployeeReactor 实例。

```
AsdkDbEmployeeReactor* m_pAsdkDbEmployeeReactor;
```

7.5.4 创建数据库反应器的实例

在收到 kLoadDwgMsg 消息的时候，给每个图形文件创建数据库反应器的实例。为了接受 kLoadDwgMsg 消息，在 CStep07App 类中实现 kLoadDwgMsg()函数。使用 Autodesk 类管理器，右键 AcRxArxApp 类的 kLoadDwgMsg()方法的节点可以很容易地实现。

7.5.5 创建 AsdkDbEmployeeReactor 对象

在 OnkLoadDwgMsg()消息函数中，创建一个新的 AsdkDbEmployeeReactor 对象：

```
DocVars.docData().m_pAsdkDbEmployeeReactor = new
        AsdkDbEmployeeReactor(acdbHostApplicationServices()->workingDatabase())
```

尝试自己实现所有其他的函数。可参考 ObjectARX 在线帮助和 Step07 项目。

7.6 测试应用

为了测试方便，像 Step03 节类似，在项目中加入 utilities.cpp 和 utilities.h 文件，并在 Step07 中加入 Create 命令。

加载本应用，用 Create 命令创建 EMPLOYEE 块。

用 Acad 的命令插入几个 EMPLOYEE 块引用。

尝试移动这些 EMPLOYEE 块引用。注意，因为我们已经附着的临时反应器的作用，它们被移动后又立即返回原位置，并在文本窗口输出：“EMPLOYEE has been reset to its original location. ”的信息。

卸载应用并关闭图形文件。

重新打开保存的图形文件。注意，因为曾经附着的是临时反应器，EMPLOYEE 块引用现在可以移动了。重新加载应用。MOVE 命令对 EMPLOYEE 块引用立刻失去了作用。

祝贺

ObjectARX 实例教程到此结束。愿你学习愉快，ObjectARX 编程成功。