# Sparring with Autodesk® ObjectARX®—Round 2 Stepping into the Ring

Lee Ambrosius – Autodesk, Inc.

**SD6174-L** ObjectARX technology is the premier API (application programming interface) that is supported by AutoCAD on Windows and Mac OS X. If you need to access APIs that aren't exposed through the AutoLISP programming language, if you want to create applications that are optimized for large data sets, or if you need to build efficient applications for Windows and Mac OS, ObjectARX may be the most effective API for you. This lecture will help you understand the basics of ObjectARX. We will discuss which development tools you need, how to interact with AutoCAD software, and how to create a very simple "Hello World-esque" project. ObjectARX technology utilizes the C++ programming language, but this session does not require that you know C++. Prior AutoCAD software programming experience with AutoLISP programming language or Microsoft .NET is highly recommended. The second session of this series is a hands-on lab.

## Learning Objectives

At the end of this class, you will be able to:

- Get started using ObjectARX
- Create a project in Microsoft Visual Studio® 2012
- Create a custom command and a custom function that accept some basic input and display a message to the user
- Load an ObjectARX technology application into AutoCAD software

## About the Speaker

*Lee is a Principal Learning Content Developer on the AutoCAD® team at Autodesk and has been an AutoCAD user for 18 years in the fields of architecture and facilities management. He has been teaching AutoCAD users for 15 years at both the corporate and college level. Lee is best known for his expertise in programming and customizing AutoCAD-based products, and has 15+ years of experience programming with AutoLISP®, VBA, Microsoft® .NET, and ObjectARX®. Lee has written articles for AUGI® publications and white papers for Autodesk on customization. He is the co-author of the AutoCAD and AutoCAD LT 2015 Bible and the author of the AutoCAD Platform Customization series.*

***Twitter:*** *http://twitter.com/leeambrosius*

***Email:*** *lee.ambrosius@autodesk.com*

***Blog:*** *http://hyperpics.blogs.com*

# Contents

# 1 Introduction

ObjectARX is a software development library that allows you to communicate directly with AutoCAD, and is not a programming language like AutoLISP. ObjectARX requires you to have an understanding of the C++ programming language. Based on your target release of AutoCAD, you will need a specific release of Microsoft Visual C++ or Objective C installed.

For AutoCAD 2007 through 2015 on Windows, you will need to use the following releases of Visual Studio:

- **Visual Studio 2012 (Update 4)** - AutoCAD 2015
- **Visual Studio 2010** - AutoCAD 2012 through 2014
- **Visual Studio 2008** - AutoCAD 2010 through 2012
- **Visual Studio .NET 2005** - AutoCAD 2007 through 2009

**Note:** Make sure to install the latest update for Visual Studio.

Microsoft Visual Studio comes in multiple editions: Express, Professional among a few others. These handouts were created and tested using Microsoft Visual Studio 2012 Professional. You can learn about the different releases of Visual Studio by going to *http://www.microsoft.com/vs/*. If you want to use Visual Studio 2012 Express, see the Appendixes section later in this handout.

For AutoCAD 2011 through 2015 on Mac OSX, you will need to use one or more of the following tools:

- Xcode - Development environment, similar to Microsoft Visual Studio on Windows.
- Qt (pronounced “cute”) - Required to build dialog boxes, the equivalent of MFC on Windows.
- Mono - Required to compile applications developed with C# on Mac OS X instead of Objective C. Mono is an open source implementation of the Microsoft .Net Framework.

The version of the development tools required varies by the release of AutoCAD being targeted. The following table outlines the version of Xcode, Qt, or Mono you will need by AutoCAD release and Mac OS X version:

| | Mac OS X | Xcode | Qt | Mono |
|---|---|---|---|---|
| **AutoCAD 2011** | 10.6.4+ (Snow Leopard) | 3.2.5 | 4.6.3 Patched | 2.6.7_3 |
| **AutoCAD 2011 SP1** | 10.6.4+ (Snow Leopard) | 3.2.5 | 4.6.3.1 Patched | 2.6.7_3 |
| **AutoCAD** | 10.6.4+ | 3.2.5 | 4.7.2 Patched | 2.10.2_5 |

| | Mac OS X | Xcode | Qt | Mono |
|---|---|---|---|---|
| **2012** | (Snow Leopard) | | | |
| **AutoCAD 2013** | 10.7.3 (Lion) | 4.3.2+ | Built-in (4.8.1) | 2.10.5 |
| | 10.8 (Mountain Lion) | 4.4 | Built-in (4.8.1) | 2.10.5 |
| **AutoCAD 2014** | 10.8 (Mountain Lion) | 4.4 | Built-in (4.8.1) | 2.10.5 |
| **AutoCAD 2015** | 10.9 (Yosemite) | 5.0.2 | Built-in (4.8.1) | 3.2.7 |

Xcode is available for download from the Mac Dev Center at *http://developer.apple.com/devcenter/mac/index.action*.

The Qt SDK can be downloaded from various locations on the Internet, the following location lists the downloads available for the QT SDKs you might need:

- *Everywhere Qt 4.8.1 (ftp://ftp.qt.nokia.com/qt/source/qt-everywhere-opensource-src-4.8.1.zip)*
- *Carbon Qt 4.7.2 (ftp://ftp.qt.nokia.com/qt/source/qt-mac-carbon-opensource-4.7.2.dmg)*
- *Cocoa Qt 4.6.3 (ftp://ftp.qt.nokia.com/qt/source/qt-mac-cocoa-opensource-4.6.2.dmg)*

Mono is available for download from the Mono project website at *http://download.mono-project.com/archive/*.

Unlike AutoLISP, C++ and Objective C applications must be compiled to machine language prior to them being executed or in the case of ObjectARX applications being loaded into AutoCAD.

The ObjectARX API contains a vast collection of classes that define functions and data types which represent most of the features of AutoCAD. Applications developed with ObjectARX can be used to retrieve information about an open drawing, create/modify drawing objects, read/write information to or from a file, and much more.

This handout only scratches the surface of what C++, Objective C, and ObjectARX have to offer; you will not become a master in C++ or Objective C after this session but you should feel more comfortable with the syntax of the programming languages. The following table lists some of the syntax you will encounter in this lab.

| Syntax | Description and Usage of Syntax |
|---|---|
| **//** | Denotes a statement that should not be executed.<br><br>`// Creates new line` |
| **/* … */** | Denotes a series of statements that should not be executed.<br><br>`/* Defines a new command.`<br>`The new command creates a circle and a line. */` |
| **;** | Denotes the end of a statement.<br><br>`int iCnt = 0;` |
| **#include** | Imports classes and functions for use in the current code module.<br><br>`#include "arxHeaders.h"` |
| ***retType funcName* (*vars …*)**<br>**{**<br>**…**<br>**}** | Defines a function. *retType* specifies the type of data that the function will return. When a function returns a value, the *return* statement must be used.<br><br>`static void Greetings()`<br>`{`<br>`  acutPrintf(_T("\nHello AU2014!"));`<br>`}`<br><br>`int addValues (int val1, int val2)`<br>`{`<br>`  return val1 + val2;`<br>`}` |

## 2 What You Need to Get Started

Before you start creating your first ObjectARX application, should obtain the following:

- **Development Environment** – Microsoft Visual Studio 2012 is what you will need to develop applications for AutoCAD 2015 on Windows or Xcode 5.0.2 to develop applications on Mac OS X. If you need to developer ObjectARX applications for an earlier release, see the section “1 - Introduction.”

- **ObjectARX Software Development Kit (SDK)** – The ObjectARX SDK contains code samples, ObjectARX library files, and the ObjectARX documentation. Proceed to *http://www.autodesk.com/objectarx* and click the License & Download link to access the download page for the ObjectARX SDKs. Download the appropriate release and version based on the AutoCAD release and platform you will be developing for.
- **ObjectARX 2015 Wizard (Windows only)** – The ObjectARX 2015 Wizard helps to setup a default project that you can use to get started with. Proceed to *http://www.autodesk.com/developautocad* and click the ObjectARX 2015 Wizard link to download the installer to your local drive. Extract the files and run *ObjectARXWizards.msi*.
- **ObjectARX Documentation** – Based on the release of the ObjectARX SDK you download the documentation might be part of the SDK or as a separate download. On Windows, the SDK documentation is available as a CHM file in the ObjectARX install folder or can be integrated into Visual Studio using the *ObjectARX Documentation.msi* install that is installed with the ObjectARX SDK. On Mac OS X, the ObjectARX documentation is not part of the ObjectARX SDK but can be found online. The link to the online ObjectARX documentation can be found at: *http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=785550*.

### A1 Download and Install the ObjectARX SDK

See A1 at the end of the handouts for instructions on how to download and install the ObjectARX SDK.

### A2 Download and Install the ObjectARX Wizard

See A2 at the end of the handouts for instructions on how to download and install the ObjectARX Wizard.

### A3 Working with VS Express 2012 for Desktop

See A3 at the end of the handouts for instructions on how to work with VS Express 2012 for Desktop.

## 3 Access the ObjectARX Libraries

ObjectARX is made up of a number of libraries that help to group related features. A library can contain one or more one class, each class is exposed as a series of header files. A header file contains the members that make up a class; members include methods, properties, among other information that defines each class. Many of the classes that make up the ObjectARX SDK are prefixed with four letters. For example, AcDb denotes classes that are related to the drawing database.

The common classes that are part of ObjectARX are listed in the following table.

| Library Prefix | Description |
|---|---|
| **AcAp** | Application level classes used to access open documents and work with transactions. |
| **AcCm** | Color class used to work with True colors and color books. |
| **AcDb** | Graphical and non-graphical object definitions stored in a drawing, such as a block insert and block definition. |
| **AcEd** | Editor services used to register commands and work with the drawing window. |
| **AcGe** | Geometry helper classes that allow you to represent points, vectors, boundaries, and in memory objects. |
| **AcPl** | Plot engine classes used to output a drawing to a hardcopy or an electronic file. |

The ObjectARX libraries are available after you install the ObjectARX SDK. By default, the ObjectARX SDK is installed to the root of your local drive. For example, the ObjectARX 2015 SDK is installed to c:\ObjectARX 2015 on Windows.

The files that you will need to reference are present in a few different folders based on the target operating system. You will be working with the files in the *inc*, *inc-win32*, *inc-x64*, *lib-win32*, and *lib-x64* folders. The *inc* folder is the most important as it contains the header files that reflect which classes are exposed and available for you to use in your ObjectARX programs.

The following is a snippet from the *dbents.h* header file that shows the *AcDbLine* class which is used to represent a Line object in a drawing.

```
class AcDbLine: public AcDbCurve
{
public:
    AcDbLine();
    AcDbLine(const AcGePoint3d& start, const AcGePoint3d& end);
    ~AcDbLine();
    ACDB_DECLARE_MEMBERS(AcDbLine);

    DBCURVE_METHODS

    Acad::ErrorStatus getOffsetCurvesGivenPlaneNormal(
        const AcGeVector3d& normal, double offsetDist,
        AcDbVoidPtrArray& offsetCurves) const;

    AcGePoint3d         startPoint() const;
    Acad::ErrorStatus   setStartPoint(const AcGePoint3d&);
```

```
    AcGePoint3d          endPoint() const;
    Acad::ErrorStatus    setEndPoint(const AcGePoint3d&);

    double               thickness() const;
    Acad::ErrorStatus    setThickness(double);

    AcGeVector3d         normal() const;
    Acad::ErrorStatus    setNormal(const AcGeVector3d&);

protected:
    virtual Acad::ErrorStatus   subGetClassID(CLSID* pClsid) const;
};
```

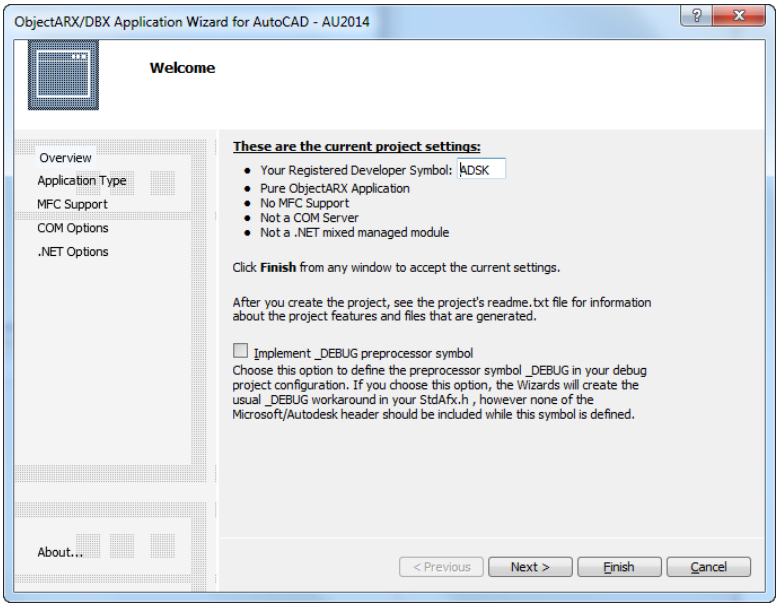
**Tip:** The *arxheaders.h* header file includes imports for many of the common classes that you will be using as part of your ObjectARX programs.

Once the libraries are installed, you must specify the location of the ObjectARX libraries in your project created with Microsoft Visual Studio 2012 and Xcode, and also import the header files into your project with the use of the *#include* statement.

## 4 Create a New Project

After Microsoft Visual C++ or Xcode, and the ObjectARX SDK have been installed, you are ready to create a new ObjectARX project. There are two ways to create a new project: from scratch or using the ObjectARX Wizard (on Windows only).

When creating a project from scratch with Microsoft Visual Studio 2012, you use the Win32 project under the Visual C++ templates. Once the project is created, you then modify the project's properties so Microsoft Visual Studio understands where the ObjectARX libraries are located and the output that the project should generate when compiled.



Using the ObjectARX Wizard does make it easier to create a new project, but it does add some extra framework that can be confusing when first getting started or might not use for your ObjectARX project. The exercises in this handout do not utilize the ObjectARX Wizard to keep the project as simple as possible.

Each ObjectARX project must include an *acrxEntryPoint* function which allows you to perform tasks when the program is loaded or unloaded. The *acrxEntryPoint* function is commonly used to add new commands and remove command groups that are defined in the ObjectARX program. The following is an example of an *acrxEntryPoint*:

```
AcRx::AppRetCode acrxEntryPoint(AcRx::AppMsgCode msg, void* appId)
{
    switch (msg) {
    case AcRx::kInitAppMsg:
        acrxDynamicLinker->unlockApplication(appId);
        acrxDynamicLinker->registerAppMDIAware(appId);

          // Add tasks here that should happen when loading the ARX file
        break;
    case AcRx::kUnloadAppMsg:
        // Add tasks here that should happen when unloading the ARX file
        break;
    }

    return AcRx::kRetOK;
}
```

In addition to the *acrxEntryPoint* function, your program must also contain a function named *acrxGetApiVersion*. However, unlike the *acrxEntryPoint* function you don't actually need to write any code for the *acrxGetApiVersion* function. To add the *acrxGetApiVersion* function to your project, you reference the *rxapi.lib* file.

### E1 Create a New Project from Scratch

See Exercises at the end of the handouts.

## 5 Compile and Load a Project

ObjectARX programs must be compiled before they can be loaded into AutoCAD. The Build menu in Microsoft Visual Studio allows you to compile a project into a DLL (Dynamic Linked Library) file that can be loaded into AutoCAD. If you configured the project correctly, the compiled file will have the extension of *.arx* on Windows or *.bundle* on Mac OS X. ObjectARX programs can be compiled for debugging or release.

ObjectARX programs that are compiled for debugging can only be ran on a computer that has the Microsoft Visual Studio installed. This is because Windows does not ship with the debug DLLs that are required to run a debugging project. Once a project has been debugged, you compile a project for release so it can be distributed and executed on other workstations.

When you compile a project, the Output window displays the location that the compiled output is stored. The Output window also displays any errors or warnings that are generated while the program is being compiled.

**Tip:** The location in which the compiled file is stored is defined by the project's properties. In Microsoft Visual Studio, the Output Directory property affects where the compiled file is stored.

The following is an example of a project that failed to compile:

```
1>------ Build started: Project: AU2014, Configuration: Debug x64 ------
1>Build started 11/17/2014 10:45:27 AM.
1>InitializeBuildStatus:
1>  Creating "x64\Debug\AU2014.unsuccessfulbuild" because "AlwaysCreate" was specified.
1>ClCompile:
1>  AU2014.cpp
1>AU2014.cpp(20): error C2001: newline in constant
1>AU2014.cpp(20): fatal error C1057: unexpected end of file in macro expansion
1>
1>Build FAILED.
1>
1>Time Elapsed 00:00:02.06
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

An ObjectARX file can be loaded into AutoCAD using a number of methods. You can use the following methods to load an ObjectARX file:

- APPLOAD command
- *arxload* AutoLISP function
- Drag and drop an ARX file into the drawing window (Windows only)
- /ld command line switch used for shortcut icons (Windows only)

### E2 Compile and Load an ObjectARX Project

See Exercises at the end of the handouts.

## 6 Define a New Command

Commands are the primary way users access features in the AutoCAD program. You expose the commands in your ObjectARX program by using the *acedRegCmds* or ACED_ARXCOMMAND_ENTRY_AUTO macro. No matter which macro you use, you must supply the following:

- Command group name that is used by AutoCAD to manage related commands
- Global or international command name; must be proceeded with an underscore when used in AutoCAD
- Local command name
- Command flags to specify the behavior of the command
- Name of the function to execute when the command name is entered

The following is an example of the syntax used to register a command with the *acedRegCmds* macro:

```
acedRegCmds->addCommand(_T("AU2014App"), _T("Greetings"),
                        _T("Hello"), ACRX_CMD_TRANSPARENT, Greetings);
```

If you are using the ACED_ARXCOMMAND_ENTRY_AUTO macro, you must specify the class that the function to execute is part of. The following is an example of the syntax used to register a command with the ACED_ARXCOMMAND_ENTRY_AUTO macro:

```
ACED_ARXCOMMAND_ENTRY_AUTO(CAU2014App, AU2014App, Greetings, Hello,
                           ACRX_CMD_TRANSPARENT, NULL)
```

Command flags are used to define the behavior of the command in AutoCAD, there are two types of command flags you can use: primary and secondary. The primary command flags are used to define if a command is transparent or modal. A transparent command can be used while another command is active, such as the ZOOM or DSETTINGS commands. Modal commands cannot be used while another command is active.

**ACRX_CMD_TRANSPARENT** – Defines a command as transparent
**ACRX_CMD_MODAL** – Defines a command as modal

Some of the secondary flags that you can use are:

**ACRX_CMD_USEPICKSET** – Pick first selection is allowed
**ACRX_CMD_REDRAW** – Redraws the screen after the command is started
**ACRX_CMD_NOPERSPECTIVE** – Command is not allowed in a perspective viewpoint
**ACRX_CMD_NOTILEMODE** – Command is not allowed in Tilemode
**ACRX_CMD_NOPAPERSPACE** – Command is not allowed in paper space
**ACRX_CMD_UNDEFINED** – Command cannot be executed unless the syntax *GroupName.CommandName* is used

Multiple command flags can be specified by separating each flag with the '|' pipe character.

```
ACRX_CMD_MODAL | ACRX_CMD_USEPICKSET
```

### E3 Define a Custom Command

See Exercises at the end of the handouts.

## 7 Access AutoCAD, Objects, System Variables, and Commands

The *acdbHostApplicationServices* class provides access to the AutoCAD application object and various other services. From the *acdbHostApplicationServices* class, you can obtain the database of the current drawing with the *workingDatabase* method. The *workingDatabase* method returns an *AcDbDatabase* object.

The following snippet shows how to get the current database:

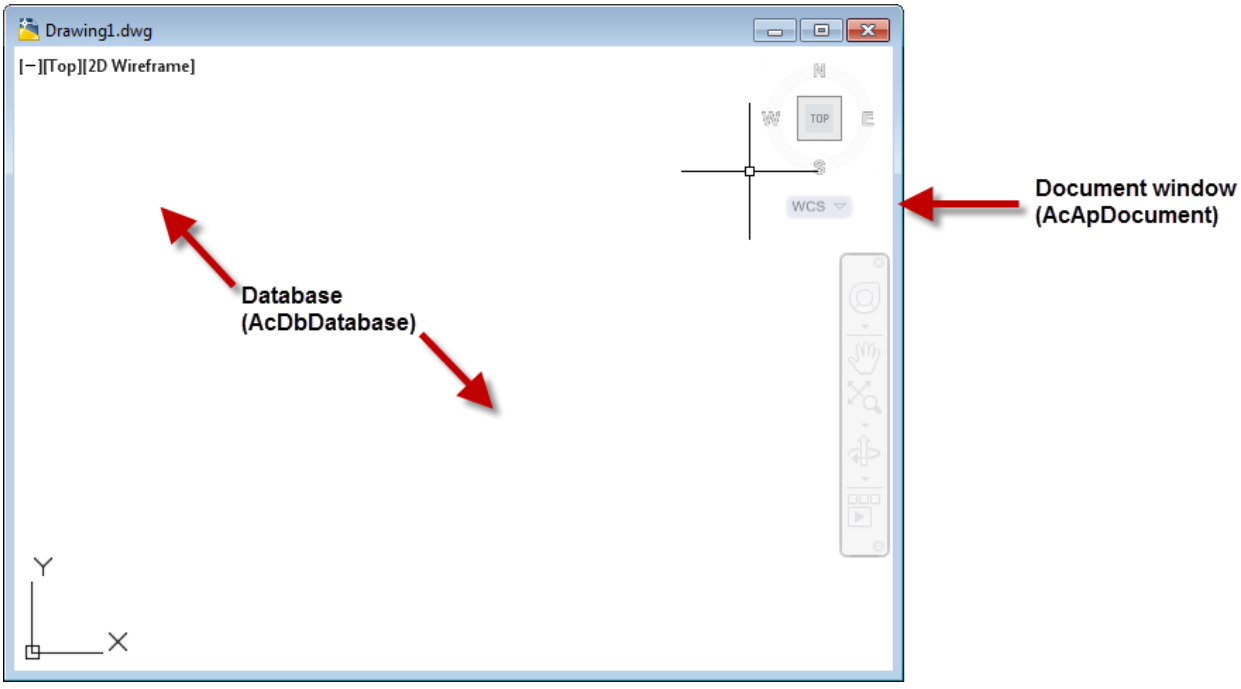
```
AcDbDatabase *pDb = acdbHostApplicationServices()->workingDatabase();
```

If you need to access other drawings than the current one, you will need to work with the Document Manager. The Document Manager can be accessed by using the *acDocManager* macro. From the Document Manager, you can open a drawing file or create a new document window (*AcApDocument* class) among other operations. Saving a drawing is done through the database object, and not the document.

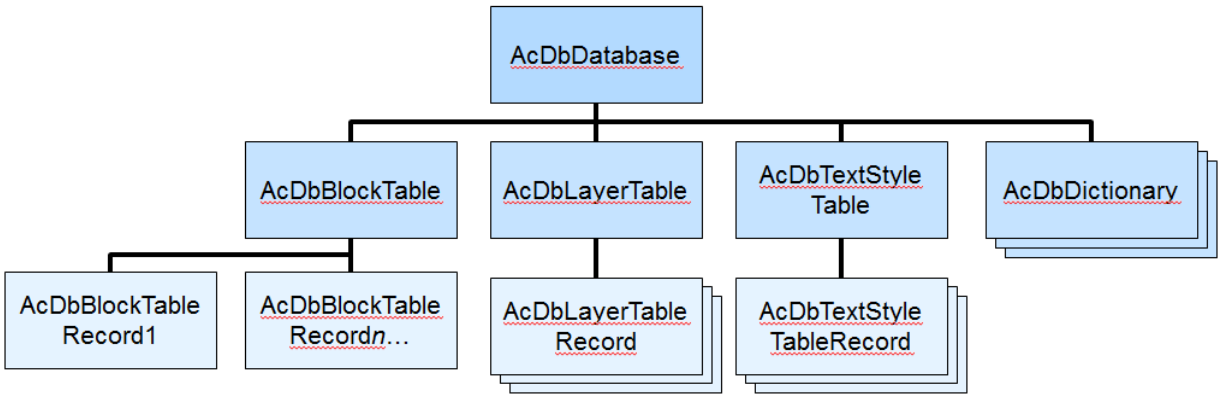
To get the database of a document, use the *database* property:

```
AcApDocument *pDoc = acDocManager->mdiActiveDocument;
AcDbDatabase *pDb = pDoc->database;
```

**Note:** The document object when it comes to ObjectARX represents the UI that contains the drawing window, and not the database that contains the objects that make up your design.



Once you have a database object, you can then create and manipulate the objects that are stored in the database. There are two types of objects in a database: graphical and non-graphical objects. Graphical objects are objects such as lines, circles, or arcs. Non-graphical objects are block definitions, text styles, and named views.




Graphical objects such as lines and circles are contained in a block table record. A block table record might be a named block that can be inserted, or model and paper space in a database. When you want to create or modify an object such as a line or circle, you will be working with one of the block table records.

The following code snippet shows how to access the model space block contained in the block table:

```
// Open the Block table for read-only
AcDbBlockTable *pBlockTable;
acdbHostApplicationServices()->workingDatabase()->
    getBlockTable(pBlockTable, AcDb::kForRead);

// Get the model space block and open it for write access
AcDbBlockTableRecord *pBlockTableRecord;
pBlockTable->getAt(ACDB_MODEL_SPACE, pBlockTableRecord, AcDb::kForWrite);
pBlockTable->close();

// Close the model space block
pBlockTableRecord->close();
```

Many of the common non-graphical objects are located in symbol tables. The following lists the symbol tables in a database:

- Block (*AcDbBlockTable*)
- Dimension style (*AcDbDimStyleTable*)
- Layer (*AcDbLayerTable*)
- Linetype (*AcDbLinetypeTable*)
- Registered applications (*AcDbRegAppTable*)

- Text style (*AcDbTextStyleTable*)
- UCSs (*AcDbUCSTable*)
- Viewports (*AcDbViewportTable*)
- Views (*AcDbViewTable*)

Other non-graphical objects introduced after AutoCAD R12 are not stored in a symbol table, but in dictionaries. At the database level, you can use one of the various get methods to access the common dictionary objects such as *getGroupDictionary* or *getLayoutDictionary*.

Once you have a block table record open for write, you can then create and append objects to it. The following shows how to create a Line and append it to model space:

```
// Define the points that will be used to define the Line object
AcGePoint3d startPt(7.0, 3.0, 0.0);
AcGePoint3d endPt(11.0, 3.0, 0.0);

// Create the new Line object in memory
AcDbLine *pLine = new AcDbLine(startPt, endPt);

// Add the new Line object to Model space
pBlockTableRecord->appendAcDbEntity(pLine);

// Close the model space block
pBlockTableRecord->close();

// Close the new line object
pLine->close();
```

When working with objects in the database, you must open an object for read or write and once done with the object it must be closed. Failing to close an object can cause AutoCAD to become unstable if the object is accessed. In one of the previous examples, it showed how to open the *AcDbBlockTable* object for read and the *AcDbBlockTableRecord* for write.

Since you are not adding a new block to the block table, opening it for read is all that needs to be done. With the model space block though, a new Line object is being added so it needs to be open for write.

Each object in a database is assigned an objectId, which is a unique index value assigned to an object as the database is loaded into memory. An object can be opened directly if you know its objectId. Objects can also be opened if you know its handle value, which unlike an objectId does not change each time the database that the object is stored in is loaded into memory. The *acdbOpenObject* method is used to open an object based on its objectId.

The following code snippet shows how to open an object using its objectId, in this case the current space:

```
// Get the current space (model or paper)
AcDbBlockTableRecord *pBlockTableRecord;

Acad::ErrorStatus es = acdbOpenObject(pBlockTableRecord,
                                      pDb->currentSpaceId(), AcDb::kForWrite);
```

When working with an object, it is recommended to open the object for read if you are not sure if you will need to modify it. If you decide you need to modify the object, you can use the *upgradeOpen* method which opens the object for write access from its original read access. You can also change an object's access from write to read by using the *downgradeOpen* method.

The following code snippet shows how to upgrade an object from read to write access:

```
// Upgrade from read access to write access
pLine->upgradeOpen();
```

System variables in AutoCAD are used to access settings that affect the behavior of the program or a command. The *acedSetVar* and *acedGetVar* methods are used to set or get the value of a system variable. You can also access some of the common system variables directly from the database object.

When using the *acedSetVar* and *acedGetVar* methods, you must work with the *resbuf* (ResultBuffer) data type which is used to control the type of data being assigned to a system variable. The *restype* property of a *resbuf* sets the type of data the result buffer can hold, and the *resval* property sets the value of the *resbuf*.

The following code snippet shows how to get and set the value for a system variable:

```
// Create a variable of the result buffer type
struct resbuf rb, rb1;

// Get the current value of CIRCLERAD
acedGetVar(_T("CIRCLERAD"), &rb);
acutPrintf(_T("\nCIRCLERAD: %.2f"), rb.resval);

// Set the value of CIRCLERAD to 2.5
rb1.restype = RTREAL;
rb1.resval.rreal = 2.5;
acedSetVar(_T("CIRCLERAD"), &rb1);
```

Commands defined by AutoCAD or those exposed by loaded ObjectARX or .NET applications can be executed from an ObjectARX application using the *acedCommandS* and *acedCommandC* methods. It is always best to recreate the functionality that you need in most cases when possible instead of relying on a command, this way your program isn't dependent

on the command. You pass pairs of parameters into the *acedCommandS* method; the data type followed by the data value.

The following lists a few of the data types that can be used to identify values of a *resbuf* or the *acedCommandS* method:

- **RTSTR** - String
- **RTREAL** - Double or real value
- **RTPOINT** - An array of three values
- **RTINT** - Integer
- **RTENAME** - Entity name
- **RTNONE** - Represents a press of the Enter key

The following code snippet shows how to execute the CIRCLE command with the *acedCommandS* method:

```
// Define the center point for the circle
ads_name pt;
pt[X] = 2.5; pt[Y] = 3.75; pt[Z] = 0.0;

// Define the radius of the circle
double rrad = 2.75;

// Execute the Circle command
if (acedCommandS(RTSTR, _T("._CIRCLE"), RTPOINT,
                 pt, RTREAL, rrad, RTNONE) != RTNORM)
{
     acutPrintf(_T("\nError: CIRCLE command failed."));
}
```

The execution of a command can be interrupted to allow a user to provide a value. To pause for input, you specify a value of PAUSE with a value of RTSTR.

The following code snippet shows how to pause for user input when using the *acedCommandS* method:

```
acedCommandS(RTSTR, _T("._CIRCLE"), RTSTR, PAUSE, RTREAL, rrad, RTNONE);
```

In addition to the *acedCommandS* method, you can also use the *sendStringToExecute* method. The *sendStringToExecute* method allows you to send a string to the command line for AutoCAD to execute, it could be to start a command or provide input to the current command.

The following code snippet shows two examples of how to send a string to the Command prompt that executes the PLINE command:

```
// Send a string to the command line for execution
acDocManager->sendStringToExecute(acDocManager->curDocument(),
                                  _T("._PLINE 0,0 5,5  "), true, false, true);
```

```
// Send an AutoLISP expression to the command line for execution
acDocManager->sendStringToExecute(acDocManager->curDocument(),
                _T("(command \"._PLINE\" PAUSE PAUSE \"\") "), true, false, true);
```

You can learn more about the *acedCommandC* method in the ObjectARX Reference Guide.

### E4 Add a Line to Model Space

See Exercises at the end of the handouts.

### E5 Create a New Layer

See Exercises at the end of the handouts.

### E6 Step through All Objects in the Database

See Exercises at the end of the handouts.

## 8 Request User Input

Most commands in AutoCAD prompt the user for one or more values. How the values are provided by the user depends on the type of information requested and the desired workflow. Input from the user in AutoCAD comes primarily in two forms: at the Command prompt or a dialog box. Input from the Command prompt is the most common form for less complex tasks. While dialog boxes are a form of input, it is beyond the scope of this session. MFC dialog boxes are commonly used with ObjectARX applications on Windows, while dialog boxes created with Qt are supported on Mac OS X.

User input methods in the ObjectARX library begin with *aced*. The following table covers some of the common input methods that allow you to request input from a user. You can learn more about these and other user input methods in the ObjectARX Reference Guide.

| Function | Description |
|---|---|
| *acedGetInt* | Pauses for an integer in the range of –32,768 and 32,767, and returns an int value. |
| *acedGetReal* | Pauses for a real or decimal number and returns an ads_real (or double) value. |
| *acedGetString* | Pauses for a string and returns a pointer to an ACHAR value. The string can contain spaces or not based on the values passed to the method. |
| *acedGetPoint* | Pauses for a coordinate value and returns an ads_point value. An ads_point represents a 3D coordinate value, and is an array of 3 double values. |

| Function | Description |
|---|---|
| *acedGetKword* | Pauses for a character string that matches a set of pre-defined options. The pre-defined options or keywords are set using the *acedInitGet* method. |

The user input methods provide some error-checking that can be used to determine if the user pressed Enter, cancelled the current operation, completed the input method, or entered a keyword. The error-checking values that the previously mentioned user input methods return are:

| Error Code | Description |
|---|---|
| RTNORM | Input entered was valid |
| RTERROR | User input method failed to complete successfully |
| RTCAN | ESC was pressed to abort the request for user input |
| RTNONE | Enter was pressed before a value was specified |
| RTREJ | The input was rejected because it was not valid |
| RTKWORD | User entered a keyword |

The *acedInitGet* method is used to specify which keywords are supported by many of the user input methods, but it is also used to control the input specified. The *acedGetString* and *acedSSGet* methods are not affected by the *acedInitGet* method.

The following shows some code snippets of requesting user input:

```
int nAge = 0;
acedGetInt(_T("\nEnter your age: "), &nAge);

TCHAR cName[30];
acedGetString(NULL, _T("\nEnter your name: "), cName);

ads_point pt;
acedGetPoint(NULL, _T("\nSpecify insertion point: "), pt);
```

When you request input from a user, it is recommended to follow the same conventions used by the standard commands in AutoCAD. Following these conventions make it natural for users to adopt the commands you implement, but it also allows for your commands to take advantage of dynamic input tooltips and contextual menus. The following explains the special character sequences used at the Command prompt:

- **< >** – Angle brackets indicate the default value that will be used when Enter is pressed without entering a value.
- **[ / ]** – Square brackets indicate the options that are available, and each option should be separated by a forward slash. The letters that a user can type in to specify an option should also appear capitalized in the prompt.

Here is an example of a prompt that uses both angle and square brackets:

```
acedInitGet(0, _T("Circle Square Hexagon"));
acedGetKword(_T("\nEnter shape [Circle/Square/Hexagon] <Circle>: "), kWord);
```

Selecting objects is another common user interaction that is used by commands to specify which object should be queried or modified in the drawing. The *acedEntSel* and *acedSSGet* methods are the most common ways of requesting the user to select an object in a drawing.

| Function | Description |
|---|---|
| *acedEntsel* | Pauses for a single object. Returns an ads_name value that contains the selected entity name and an ads_point that contains the coordinate picked in the drawing window. |
| *acedSSGet* | Pauses for a set of objects, and returns an ads_name value that contains the name of the selection set created. |

The following code snippet shows an example of using the *acedEntSel* method:

```
ads_point ePt;
ads_name eName;
acedEntSel(_T("\nSelect an entity: "), eName, ePt)
```

The following code snippet shows an example of using the *acedSSGet* method:

```
ads_name sset;
acedSSGet(NULL, NULL, NULL, NULL, sset);
```

Once you have a selection set returned from the *acedSSGet* method, you can determine the number of objects that were selected using the *acedSSLength* method and use *acedSSName* to get an object in the selection set.

The following code snippet gets the first object in the selection set:

```
ads_name ename;
acedSSName(sset, 0, ename);
```

After you get an entity from a selection set, you use the *acdbGetObjectId* method to get the objectId of the entity and *acdbOpenObject* method to open the object so it can be read or modified.

In addition to requesting input from a user, you can use the *acutPrintf* and *acedAlert* methods to provide information to the user at the Command prompt or in a message box. The *acutPrintf* method allows you to display small amounts of information to the user and not have it get in the way of the current workflow.

If you display too much information at the Command prompt, the user has a chance of missing it. The *acedTextScr* method can be used to open the AutoCAD Text Window or expand the Command Line window so the user does not miss a message that you might have displayed to them, while the *acedGraphScr* method displays the drawing window (or graphics screen).

The *acedAlert* method slows down the current user workflow, but ensures the user is aware of a problem or an important piece of information.

The following code snippet demonstrates the *acedAlert* method:

```
acedAlert(_T("An error occurred!"));
```

The *acedAlert* method is rather limited in functionality. You can also use the *AfxMessageBox* function of the C++ programming language on Windows or the NSAlert class on Mac OS X if you need additional control over the appearance of the message box.

### E7 Add a Line using User Input

See Exercises at the end of the handouts.

### E8 Select Objects and Request a Keyword

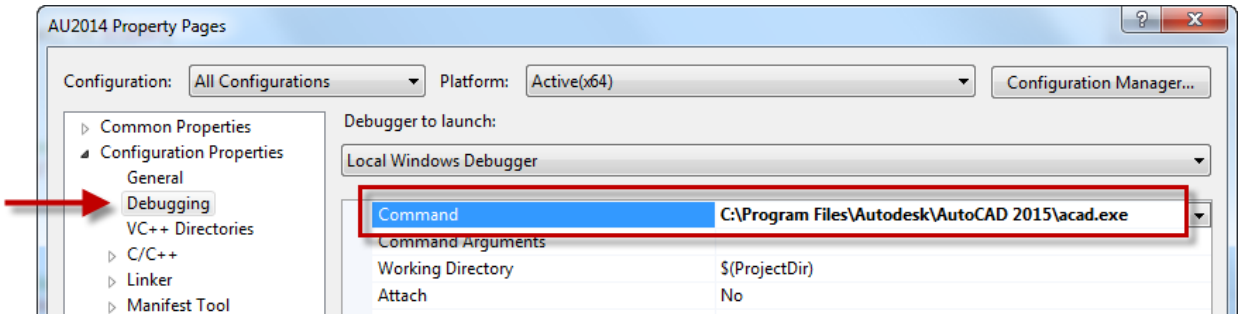
See Exercises at the end of the handouts.

## 9 Debugging a Project

Microsoft Visual Studio and Xcode offer the ability to debug your applications in real-time while it is loaded in AutoCAD. You can set breakpoints and then step through your ObjectARX program line by line while it is being executed. This type of debugging is much more efficient than putting random print or message statements to try and figure out where and what problems exist in a program. Using the development environment, you can also see the current values assigned to a variable.

ObjectARX does not provide any additional debug tools, you simply use those built into Microsoft Visual Studio and Xcode. On Windows, you can use utilities/add-ons like Bounds Checker to help perform advanced debugging and memory leak detection.

In Microsoft Visual Studio, when you want to debug your ObjectARX application, you must set the Command property under the Debugging category to the *acad.exe*.



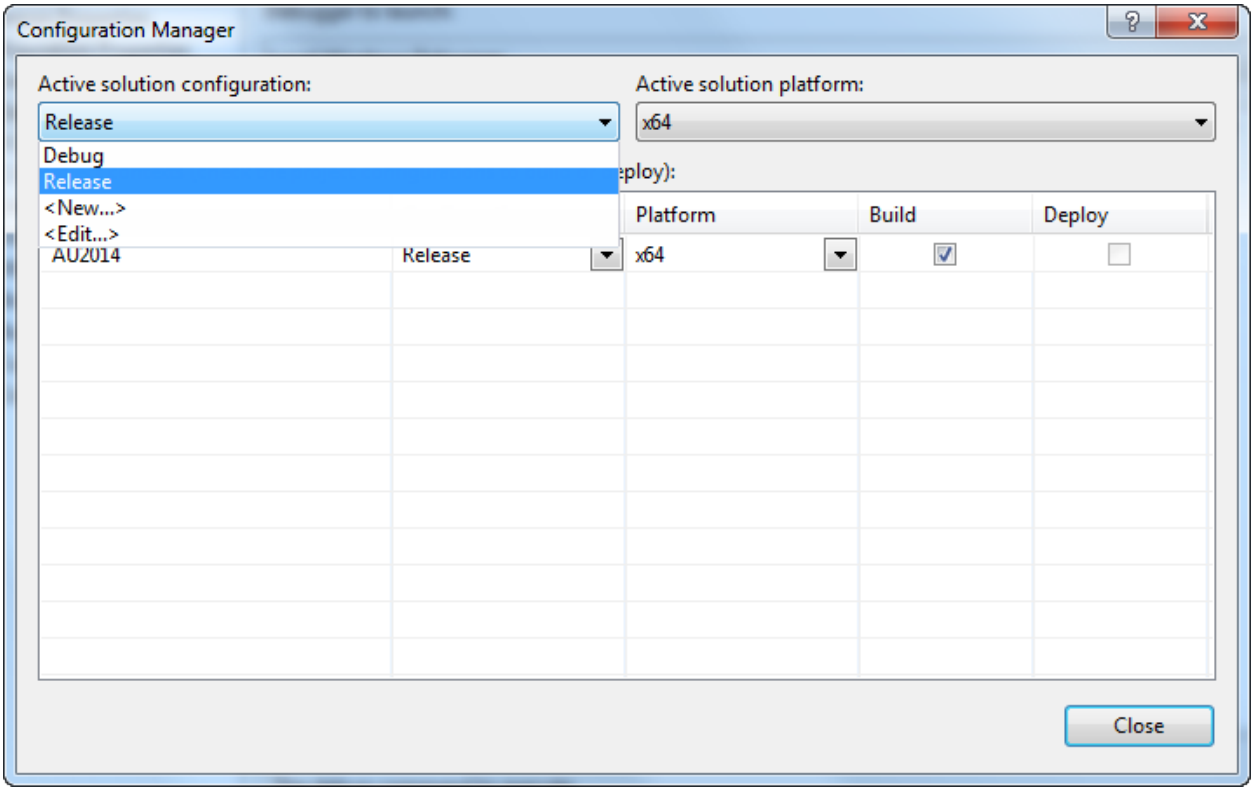

## E9 Debug an ObjectARX Application

See Exercises at the end of the handouts.

# 10 Build a Project for Release

When building an ObjectARX project, by default it is compiled for debugging which includes additional information that allows access to the debugger environment in Microsoft Visual Studio or Xcode while loaded into AutoCAD. A project compiled for debugging can only be used on a workstation that contains all the debugging DLLs, which are commonly only installed with Microsoft Visual Studio.

For the project to be accessible on other workstations that do not contain the developer environment, you must compile the project for release. In Microsoft Visual Studio, switching between Debug and Release configurations is handled through the Active Solution Configuration drop-down list in Configuration Manager. The settings for the configuration should match the values mentioned in E1 Create a New Project from Scratch.



### E10 Build an ObjectARX Application for Release

See Exercises at the end of the handouts.

## 11 Where to Get More Information

When you are first starting to use a new feature, you will have questions and where you go to find answers might not be clear. The following is a list of resources that you can use to get help:

- **Microsoft Visual Studio** – Visual Studio is the development platform that Microsoft develops and supports for Windows development. You can locate information on Visual Studio and Windows development at *http://www.microsoft.com/vs/* and *http://msdn.microsoft.com/*.
- **C++** –C++ is the premier programming language used for Windows development. The programming language is object orientated and derived from C. You can learn about C++ at *http://msdn.microsoft.com/en-us/library/windows/desktop/ff381399(v=vs.85).aspx*. You can also learn about Objective-C with books such as C++ Primer published by Addison-Wesley Professional, or online training websites such as *lynda.com*.
- **Xcode and Apple Development** – Xcode is the development platform used to create applications on Mac OS X. You can locate information on Xcode and Mac OS X development at *https://developer.apple.com/xcode/downloads/*.
- **Objective-C** – Objective-C is the primary programming language used for Mac OS X development. The programming language is object orientated, and derived from C and C++. You can learn about Objective-C at *https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html*. You can also learn about Objective-C with books such as Programming in Objective-C published by Pearson P T R, or online training websites such as *lynda.com*.
- **Help System** – Help System – The ObjectARX Reference (arxref.chm) and ObjectARX Developer Guides (*arxdev.chm*) in the ObjectARX SDK install folder contains a lot of information on working with ObjectARX. To access the help docs go to *C:\ObjectARX 2015\docs*.
- **Autodesk Discussion Forums** – The Autodesk forums provide for peer-to-peer networking and some interaction with Autodesk moderators. You can ask a question about anything in AutoCAD and get a response from a fellow user or Autodesk employee. To access the forum discussions about ObjectARX, go to *http://forums.autodesk.com*, click AutoCAD, and then click Autodesk ObjectARX.
- **AUGI Forums** – The AUGI forums provide peer-to-peer networking where you can ask questions about virtually anything in AutoCAD and get a response from a fellow user. Visit AUGI at *http://www.augi.com/*.
- **Blogs** – There are a few blogs out there that cover ObjectARX development and they are:
  - **AutoCAD DevBlog** - *http://adndevblog.typepad.com/autocad/*
  - **Through the Interface** - *http://through-the-interface.typepad.com/*

- **ObjectARX & Dummies** - *http://arxdummies.blogspot.com/*

- **Industry Events and Classes** – Industry events such as AUGI CAD Camp and Autodesk University are great places to learn about new features in an Autodesk product. Along with industry events, you might also be able to find classes at your local technical college or Autodesk Authorized Training Center (ATC).
- **Internet** – There are tutorials on the Internet to learn many of the customization and programming options that are offered in AutoCAD. Use your favorite search engine, such as Google or Bing and search on the topic of interest.

# 12 Exercises

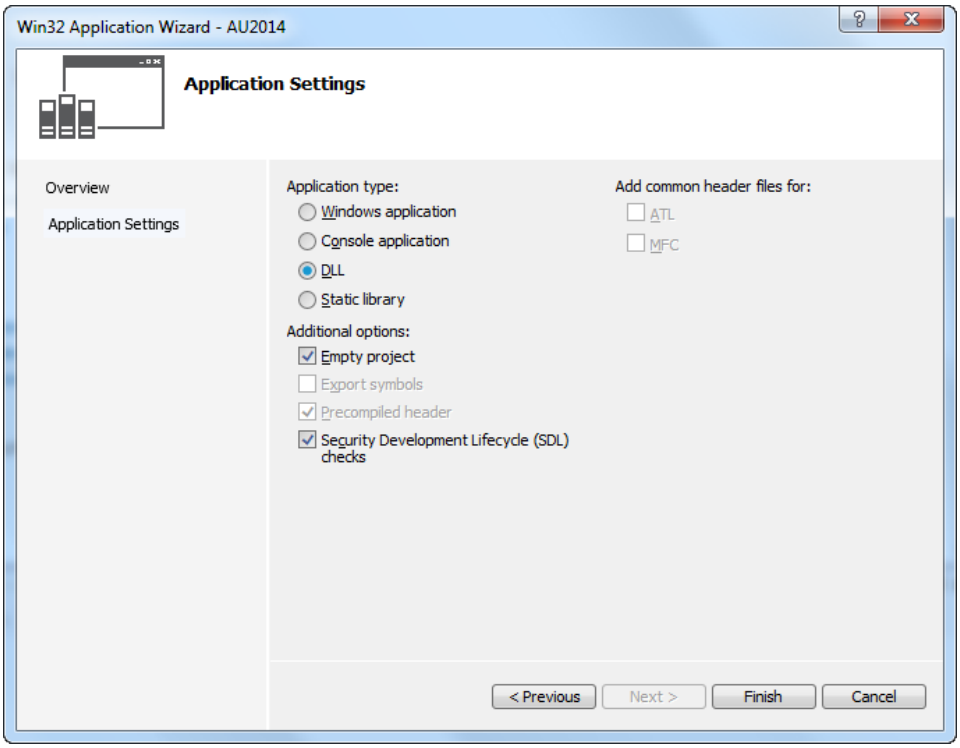
This section contains all the exercises for this lab and for when you get back to the office.

## E1 Create a New Project from Scratch (Microsoft Visual Studio 2012)

This exercise explains how to create a new Microsoft Visual C++ project that can be used to create and compile an ObjectARX project.

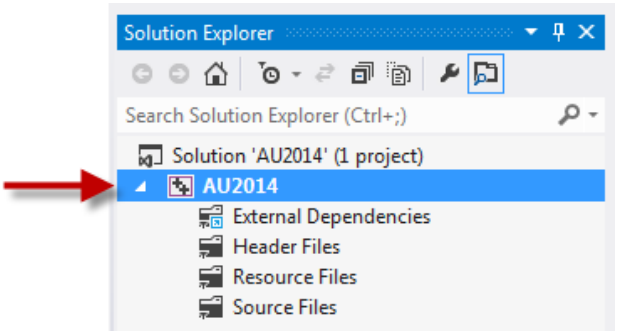
1. Click Start ➢ [All] Programs ➢ Microsoft Visual Studio 2012 ➢ Visual Studio 2012.
2. In Microsoft Visual Studio, on the menu bar, click File ➢ New ➢ Project.
3. In the New Project dialog box, under Installed ➢ Templates, select Visual C++ and then click the .NET Framework drop-down list along the top and choose .NET Framework 4.5.
4. In the Templates list, select Win32 Project.
5. In the Name field, select the current value and type **AU2014**.
6. Click Browse to the right of the Location field.
7. In the Project Location dialog box, browse to and select the folder for this session. Click Select Folder.

   The name of the session folder should be
   *C:\Datasets\Wednesday\SD6174-L Sparring with Autodesk® ObjectARX® - Round 2 Stepping into the Ring*
8. In the New Project dialog box, click OK.
9. In the Win32 Application Wizard - AU2014 dialog box, click Next.
10. On the Application Settings page, under Application Type, select DLL.
11. Under Additional Options, check Empty Project and click Finish.

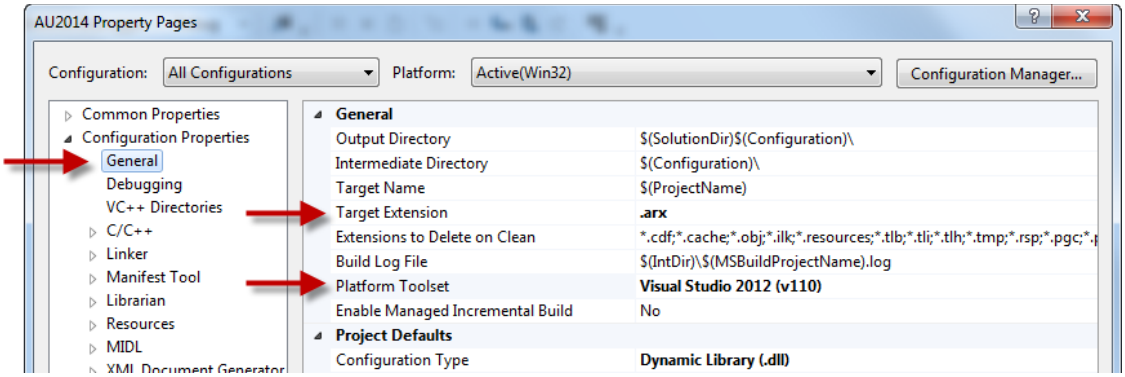



Now that you have a project, you can configure its settings to specify where the ObjectARX libraries can be found and other build settings.

1. In the Solution Explorer, select AU2014. Right-click and choose Properties.




2. In the Property Pages dialog box, expand Configuration Properties and select General.
    a. From the Configuration drop-down list, select All Configurations.
    b. Click the Target Extension field and change **.dll** to **.arx**.
    c. Click the Platform Toolset field, click the drop-down arrow, and choose **Visual Studio 2012 (V110)**.

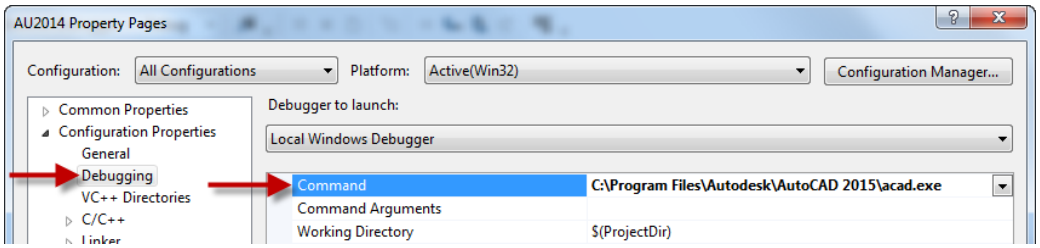



3. Under Configuration Properties, select Debugging.

    a. Click in the Command field.

    b. Click the drop-down arrow, and choose <Browse…>.

    c. Browse to C:\Program Files\Autodesk\AutoCAD 2015 and select acad.exe. Click Open.




4. Under Configuration Properties and select VC++ Directories.

    a. Click in the Include Directories field.

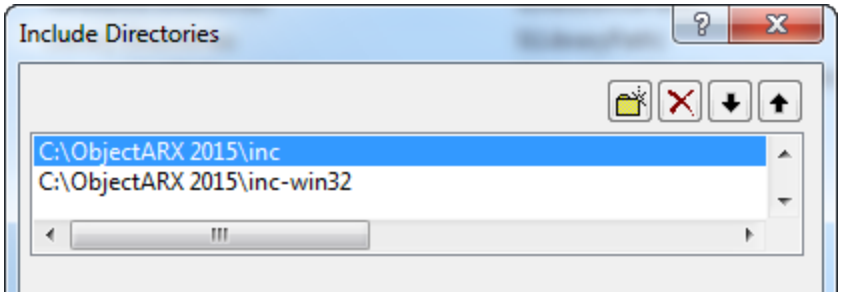
    b. Click the drop-down arrow, and choose <Edit…>.

    c. In the Include Directories dialog box, click New Line (Folder icon) and then click the Ellipsis button.

    d. In the Select Directory dialog box, browse to and select C:\ObjectARX 2015\inc-win32. Click Select Folder.

    e. Click New Line again and then click the Ellipsis button.

    f. In the Select Directory dialog box, browse to and select C:\ObjectARX 2015\inc. Click Select Folder.



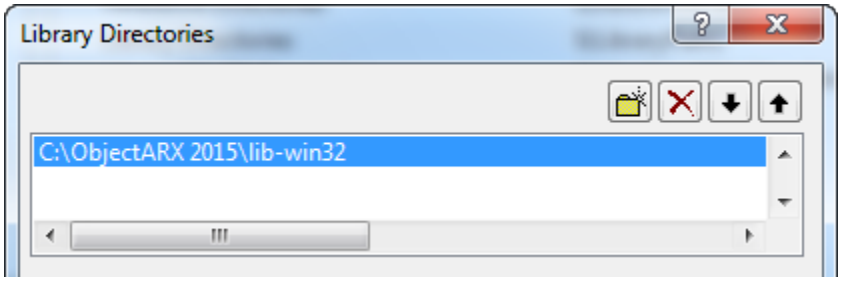

g. Click OK to return to the Property Pages dialog box.

h. Click in the Library Directories field.

i. Click the drop-down arrow, and choose <Edit…>.

j. In the Library Directories dialog box, click New Line and then click the Ellipsis button.

k. In the Select Directory dialog box, browse to and select C:\ObjectARX 2015\lib-win32. Click Select Folder and then click OK.



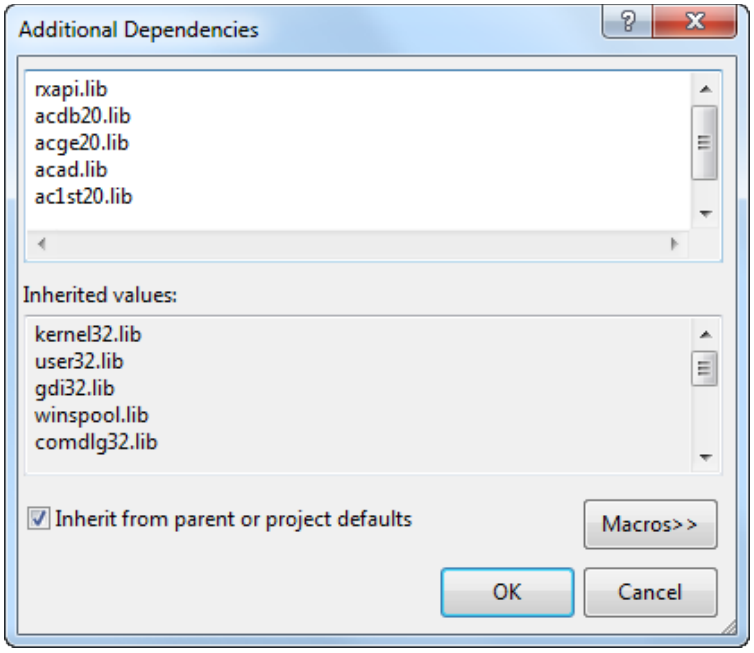

5. Under Configuration Properties, expand Linker and select Input.

a. Click in the Additional Dependencies field.

b. Click the drop-down arrow, and choose <Edit…>.

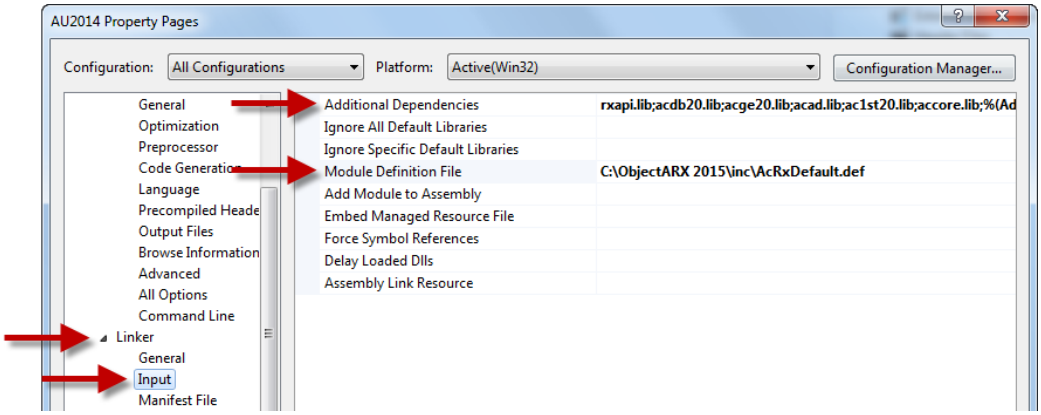
c. In the Additional Dependencies dialog box, enter the following on separate lines:

- **rxapi.lib**
- **acdb20.lib**
- **acge20.lib**
- **acad.lib**
- **ac1st20.lib**
- **accore.lib**




d. Click OK.

e. Click in the Module Definition File field and type **C:\ObjectARX 2015\inc\AcRxDefault.def**.




6. Under Configuration Properties, expand C/C++ and select Code Generation.

   a. Click in the Runtime Library field.

   b. Click the drop-down arrow, and choose Multi-threaded DLL (/MD).

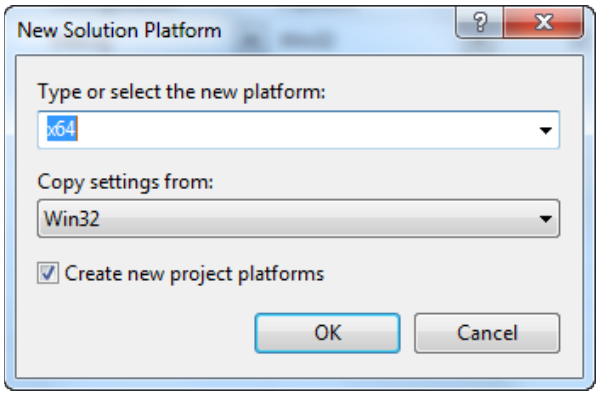
   **Note:** Don't do this as part of the session. Optionally, expand C/C++ and select General. Click in the Warning Level field and choose Level1 (/W1) from the drop-down list.

7. Click Apply.

8. Along the top of the Property Pages dialog box, click the Configuration drop-down list and select Debug.
    a. Under Configuration Properties, C/C++, select Preprocessor.
    b. Click in the Preprocessor Definitions field.
    c. Click the drop-down arrow, and choose <Edit…>.
    d. In the Preprocessor Definitions dialog box, change _DEBUG to NDEBUG and click OK.
9. Click Apply.

Now that you have configured the project for the Win32 platform, you must configure the project for the x64 platform. Win32 is used for 32-bit releases of AutoCAD run on Windows, while x64 is used for 64-bit releases of AutoCAD run on Windows.

1. In the Property Pages dialog box, click Configuration Manager.
    a. In the Configuration Manager, click the Active Solution Platform drop-down list and choose <New…>.
    b. In the New Solution Platform dialog box, click the Type or Select the New Platform drop-down list and choose x64.
    c. Check Create New Project Platforms if it isn't already checked and click OK.




    d. In the Configuration manager, click Close to return to the Property Pages.
2. Along the top of the Property Pages dialog box, click the Configuration drop-down list and select All Configurations.
    a. Under Configuration Properties, select VC++ Directories.
    b. Click in the Include Directories field.

c. Click the drop-down arrow, and choose <Edit…>.

d. In the Include Directories dialog box, double-click the C:\ObjectARX 2015\inc-win32 path and change inc-win32 to inc-x64. Click OK.

e. Click in the Library Directories field.

f. Click the drop-down arrow, and choose <Edit…>.

g. In the Library Directories dialog box, double-click the C:\ObjectARX 2015\lib-win32 path twice and change lib-win32 to lib-x64. Click OK.

h. Click Apply.

3. Click OK to save the changes to the project's properties.

The following explains how to add a source file to the project. This source file is where you will add any new commands and functions that define your program.

1. In Microsoft Visual Studio 2012, on the menu bar, click Project ➢ Add New Item.
2. In the Add New Item dialog box, under Installed, select Visual C++ and then select C++ File (.cpp).
3. In the Name field, select the current value and type **AU2014**. Click Add.
4. On the Windows taskbar, right-click the Start button and choose Open Windows Explorer.
5. In Windows Explorer, browse to *C:\Datasets\Wednesday\SD6174-L Sparring with Autodesk® ObjectARX® - Round 2 Stepping into the Ring* and double-click *E1 - Sample Code.txt* to open the file in Notepad.
6. In Notepad, press Ctrl+A and then press Ctrl+C.
7. Switch back to Microsoft Visual Studio 2012.
8. In Microsoft Visual Studio, click in the code editor window and press Ctrl+V.

   The code editor window should now look like the following.

```
// Load the common Windows headers
#include <windows.h>

// Load the common AutoCAD headers
#include "arxHeaders.h"
#include "dbents.h"

extern "C" AcRx::AppRetCode acrxEntryPoint(AcRx::AppMsgCode msg, void* pkt)
```

```
{
    switch(msg)
    {
        case AcRx::kInitAppMsg:
            acrxDynamicLinker->unlockApplication(pkt);
            acrxDynamicLinker->registerAppMDIAware(pkt);

            acutPrintf(_T("\nLoading AU2014 project...\n"));

            break;
        case AcRx::kUnloadAppMsg:
            acutPrintf(_T("\nUnloading AU2014 project...\n"));

            break;
        default:
            break;
    }
    return AcRx::kRetOK;
}
```

9. On the menu bar, click File ➢ Save All.

## E2 Compile and Load an ObjectARX Project

This exercise explains how to compile the AU2014 ObjectARX project and then load it into AutoCAD.

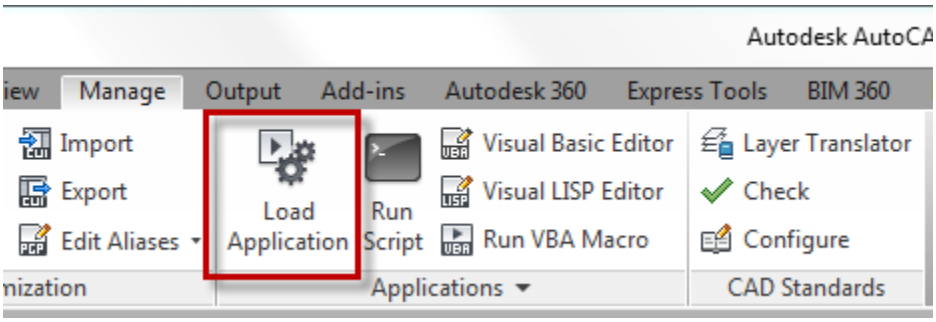
1. In Microsoft Visual Studio, on the menu bar, click Build ➢ Build Solution.

   The location of the build output is displayed in the Output window.

   ```
   1>Link:
   1>  AU2014.vcxproj -> C:\Datasets\Wednesday\SD6174-L Sparring with
   Autodesk® ObjectARX® - Round 2 Stepping into the
   Ring\AU2014\x64\Debug\AU2014.arx
   1>FinalizeBuildStatus:
   1>  Deleting file "x64\Debug\AU2014.unsuccessfulbuild".
   1>  Touching "x64\Debug\AU2014.lastbuildstate".
   1>
   1>Build succeeded.
   ```

2. Start AutoCAD 2015.

3. In AutoCAD, on the ribbon, click Manage tab ➢ Applications panel ➢ Load Application.




4. In the Load/Unload Applications dialog box, and browse to the location of the compiled ARX file. Select AU2014.arx and click Load.
5. In the File Loading - Security Concern message box, click Load.
6. Click Close.
7. Press F2 to display the AutoCAD Text Window or expand the Command Line window. You should notice that the following messages have been displayed.

```
Loading AU2014 project...
AU2014.arx successfully loaded.
```

The message 'Loading AU2014 project...' is displayed using the *acutPrintf* method in the *acrxEntryPoint* function when *kInitAppMsg* is handled.

Once an ObjectARX application is loaded, you cannot recompile it until you unload the compiled output file. The following steps explain how to unload an ObjectARX application from AutoCAD.

1. In AutoCAD, on the ribbon, click Manage tab ➢ Applications panel ➢ Load Application.
2. In the Load/Unload Applications dialog box, under the Loaded Applications tab, select AU2014.arx and click Unload.
3. Click Close.
4. Press F2 to display the AutoCAD Text Window or expand the Command Line window. You should notice that the following messages have been displayed.

```
Unloading AU2014 project...
au2014.arx successfully unloaded.
```

The message 'Unloading AU2014 project...' is displayed using the *acutPrintf* method in the *acrxEntryPoint* function when *kUnloadAppMsg* is handled.

## E3 Define a Custom Command

This exercise explains how to create a basic function and define a custom command.

1. In Microsoft Visual Studio, place the cursor after the line `#include "dbents.h"` in the source code file and press Enter twice.

2. Enter the following code to define a function named *Greetings* which displays a message at the Command prompt.

```
static void Greetings()
{
    acutPrintf(_T("\nHello AU2014!!!"));
}
```

3. Now that a function exists, you can add the code to define the command in AutoCAD and remove the command when the ObjectARX application is unloaded. Add the following lines in bold to the *acrxEntryPoint* function. You do not need to enter the lines of code that start with `//`.

```
extern "C" AcRx::AppRetCode acrxEntryPoint(AcRx::AppMsgCode msg, void* pkt)
{
    switch(msg)
    {
        case AcRx::kInitAppMsg:
            acrxDynamicLinker->unlockApplication(pkt);
            acrxDynamicLinker->registerAppMDIAware(pkt);

            acutPrintf(_T("\nLoading AU 2014 project...\n"));

            // Commands to add
            acedRegCmds->addCommand(_T("AUCommands"), _T("First"),
                                    _T("Uno"), ACRX_CMD_MODAL, Greetings);

            break;
        case AcRx::kUnloadAppMsg:
            acutPrintf(_T("\nUnloading AU 2014 project...\n"));

            // Command Groups to remove
            acedRegCmds->removeGroup(_T("AUCommnds"));

            break;
        default:
            break;
    }
    return AcRx::kRetOK;
}
```

4. Save the changes to the ObjectARX project.

5. In Microsoft Visual Studio, on the menu bar, click Build ➢ Build Solution.

   If you see the following error in the Output window, the ARX file was not unloaded from AutoCAD.

```
1>  AU2014.cpp
1>LINK : fatal error LNK1168: cannot open C:\Datasets\Wednesday\SD6174-L
Sparring with Autodesk® ObjectARX® - Round 2 Stepping into the
Ring\AU2014\x64\Debug\AU2014.arx for writing
1>
1>Build FAILED.
1>
1>Time Elapsed 00:00:03.05
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

6. If the build failed, make sure AU2014.arx is unloaded from AutoCAD.

   a. In AutoCAD, on the ribbon, click Manage tab ➢ Applications panel ➢ Load Application.

   b. In the Load/Unload Applications dialog box, under the Loaded Applications tab, select AU2014.arx and click Unload.

7. In AutoCAD, on the ribbon, click Manage tab ➢ Applications panel ➢ Load Application.

8. In the Load/Unload Applications dialog box, and browse to the location of the compiled ARX file. Select AU2014.arx and click Load.

   If the File Loading - Security Concern message box is displayed, click Load.

9. Click Close.

10. At the Command prompt, type **uno** and press Enter.

    The message “Hello AU2014!!!” is displayed. If you don’t see the message, press F2 to display the AutoCAD Text Window or expand the Command Line window.

11. At the Command prompt, type **first** and press Enter.

    You should notice that the command does not work because it is the global/international command name and requires the use of an underscore to be placed in front of the command name. What might happen is that the PICKFIRST system variable is started.

12. If the PICKFIRST system variable is active, press Esc to get to a clean Command prompt.

13. At the Command prompt, type _**first** and press Enter.

    The command should work as expected and display the message “Hello AU 2014!!!”.

14. Unload the AU2014.arx file from AutoCAD.

## E4 Add a Line to Model Space

This exercise explains how to create a new graphical object and at it to model space, in this case a Line object.

1. In Microsoft Visual Studio, place the cursor after the line `#include "dbents.h"` in the source code file and press Enter twice.

2. Type the following code to define a function named *addLine* which adds a new Line object to the Model space block. You do not need to enter the lines of code that start with `//`.

```
static void addLine()
{
    // Get the current database
    AcDbDatabase *pDb = acdbHostApplicationServices()->workingDatabase();

    // Open the Block Table for read-only
    AcDbBlockTable *pBlockTable;
    pDb->getSymbolTable(pBlockTable, AcDb::kForRead);

    // Get the Model Space block
    AcDbBlockTableRecord *pBlockTableRecord;
    pBlockTable->getAt(ACDB_MODEL_SPACE,
                       pBlockTableRecord, AcDb::kForWrite);
    pBlockTable->close();

    // Define the points that will be used to define the Line object
    AcGePoint3d startPt(7.0, 3.0, 0.0);
    AcGePoint3d endPt(11.0, 3.0, 0.0);

    // Create the new Line object in memory
    AcDbLine *pLine = new AcDbLine(startPt, endPt);

    // Add the new Line object to Model space
    pBlockTableRecord->appendAcDbEntity(pLine);

    // Close the Model space block
    pBlockTableRecord->close();

    // Close the new line object
    pLine->close();
}
```

3. In the *acrxEntryPoint* function, make the following addition in bold.

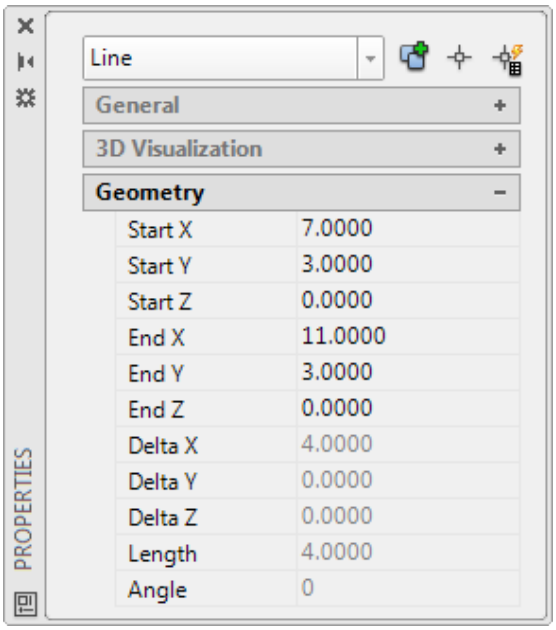
```
// Commands to add
acedRegCmds->addCommand(_T("AUCommands"), _T("First"),
                        _T("Uno"), ACRX_CMD_MODAL, Greetings);
```

```
acedRegCmds->addCommand(_T("AUCommands"), _T("AddLine"),
                        _T("AddLine"), ACRX_CMD_MODAL, addLine);
```

4. Save the changes to the ObjectARX project.
5. Build the ObjectARX project and load the *AU2014.arx* file into AutoCAD.

   Don't forget to unload the *AU2014.arx* file from AutoCAD or close and restart AutoCAD before building the project.
6. At the Command prompt, type **addline** and press Enter.
7. Zoom to the extents of the drawing to see the whole.
8. In the drawing area, select the line. Right-click and choose Properties.

   You should notice that it has a start point of 7,3 and an end point of 11,3.




9. Close the Properties palette and unload the AU2014.arx file from AutoCAD.

## E5 Create a New Layer

This exercise explains how to create a non-graphical object, in this case a Layer. Blocks and other non-graphical objects can be added using the same process.

1. In Microsoft Visual Studio, place the cursor after the line `#include "dbents.h"` in the source code file and press Enter twice.
2. Type the following code to define a function named *makeLayer* which adds a new Layer to the Layer table object. You do not need to enter the lines of code that start with `//`.

```
static void makeLayer()
{
    // Open the Layer table for read
    AcDbDatabase *pDb = acdbHostApplicationServices()->workingDatabase();
    AcDbLayerTable *pLayerTable;
    pDb->getLayerTable(pLayerTable, AcDb::kForRead);

    // Check to see if the layer exists
    if (pLayerTable->has(_T("OBJ")) == false)
    {
        // Open the Layer table for write
        pLayerTable->upgradeOpen();

        // Create the new layer and assign it the name 'OBJ'
        AcDbLayerTableRecord *pLayerTableRecord =
            new AcDbLayerTableRecord();
        pLayerTableRecord->setName(_T("OBJ"));

        // Set the color of the layer to cyan
        AcCmColor color;
        color.setColorIndex(4);
        pLayerTableRecord->setColor(color);

        // Add the new layer to the Layer table
        pLayerTable->add(pLayerTableRecord);

        // Close the Layer table and record
        pLayerTable->close();
        pLayerTableRecord->close();
    }
}
```

3. In the *acrxEntryPoint* function, make the following addition in bold.

```
acedRegCmds->addCommand(_T("AUCommands"), _T("AddLine"),
                        _T("AddLine"), ACRX_CMD_MODAL, addLine);
acedRegCmds->addCommand(_T("AUCommands"), _T("MakeLayer"),
                        _T("MakeLayer"), ACRX_CMD_MODAL, makeLayer);
```

4. Save the changes to the ObjectARX project.
5. Build the ObjectARX project and load the *AU2014.arx* file into AutoCAD.
6. At the Command prompt, type **makerlayer** and press Enter.
7. From the ribbon, on the Home tab ➢ Layers panel click the Layer drop-down list. The new layer named OBJ should now be listed.

   The OBJ layer should be assigned the color Cyan.
8. Unload the AU2014.arx file from AutoCAD.

## E6 Step through All Objects in the Database

This exercise explains how to step through all the objects in Model space without prompting the user for a set of objects.

1. In Microsoft Visual Studio, place the cursor after the line `#include "dbents.h"` in the source code file and press Enter twice.

2. Type the following code to define a function named *listObjects* which displays the types of objects in model space. You do not need to enter the lines of code that start with `//`.

```
static void listObjects()
{
    // Get the current database
    AcDbDatabase *pDb = acdbHostApplicationServices()->workingDatabase();

    // Get the current space object
    AcDbBlockTableRecord *pBlockTableRecord;
    Acad::ErrorStatus es = acdbOpenObject(pBlockTableRecord,
                                          pDb->currentSpaceId(),
                                          AcDb::kForRead);

    // Create a new block iterator that will be used to
    // step through each object in the current space
    AcDbBlockTableRecordIterator *pItr;
    pBlockTableRecord->newIterator(pItr);

    // Create a variable AcDbEntity type which is a generic
    // object to represent a Line, Circle, Arc, among other objects
    AcDbEntity *pEnt;

    // Step through each object in the current space
    for (pItr->start(); !pItr->done(); pItr->step())
    {
        // Get the entity and open it for read
        pItr->getEntity(pEnt, AcDb::kForRead);

        // Display the class name for the entity before
        // closing the object
        acutPrintf(_T("\nClass name: %s"), pEnt->isA()->name());
        pEnt->close();
    }

    // Close the current space object
    pBlockTableRecord->close();

    // Remove the block iterator object from memory
    delete pItr;

    // Display the AutoCAD Text Window
    acedTextScr();
}
```

3. In the *acrxEntryPoint* function, make the following addition in bold.

```
acedRegCmds->addCommand(_T("AUCommands"), _T("MakeLayer"),
                        _T("MakeLayer"), ACRX_CMD_MODAL, makeLayer);
acedRegCmds->addCommand(_T("AUCommands"), _T("ListObjects"),
                        _T("ListObjects"), ACRX_CMD_MODAL, listObjects);
```

4. Save, build, and load the *AU2014.arx* file into AutoCAD.
5. Add some objects to the current drawing, such as lines, circles, polylines, and so on.
6. At the Command prompt, type **listobjects** and press Enter.
7. Close the AutoCAD Text Window or collapse the Command Line window when done.
8. Unload the AU2014.arx file from AutoCAD.

## E7 Add a Line Using User Input

This exercise explains how to use the *acedGetPoint* function to request points in the drawing and then apply the input to draw a Line object in the current space.

1. In Microsoft Visual Studio, place the cursor after the line `#include "dbents.h"` in the source code file and press Enter twice.
2. Type the following code to define a function named *inputLine* which allows you to draw a line based on points specified by the user in the drawing area. You do not need to enter the lines of code that start with `//`.

```
static void inputLine()
{
    // Get the current space block
    AcDbDatabase *pDb = acdbHostApplicationServices()->workingDatabase();
    AcDbBlockTableRecord *pBlockTableRecord;
    Acad::ErrorStatus es = acdbOpenObject(pBlockTableRecord,
                                          pDb->currentSpaceId(),
                                          AcDb::kForWrite);

    // Create 2 variables of the old point data type
    ads_point pt1, pt2;

    // Prompt for the first point
    if (RTNORM == acedGetPoint(NULL, _T("\nSpecify first point: "), pt1))
    {
        AcGePoint3d startPt(pt1[0], pt1[1], pt1[2]);

        // Prompt for the second point
        if (RTNORM == acedGetPoint(pt1,
                                   _T("\nSpecify second point: "), pt2))
        {
```

```
            AcGePoint3d endPt(pt2[0], pt2[1], pt2[2]);

            // Create and append the new Line object
            AcDbLine *pLine = new AcDbLine(startPt, endPt);
            pBlockTableRecord->appendAcDbEntity(pLine);

            // Close the block table record and the Line object
            pBlockTableRecord->close();
            pLine->close();
        }
    }
}
```

3. In the *acrxEntryPoint* function, make the following addition in bold.

```
acedRegCmds->addCommand(_T("AUCommands"), _T("ListObjects"),
                        _T("ListObjects"), ACRX_CMD_MODAL, listObjects);
acedRegCmds->addCommand(_T("AUCommands"), _T("InputLine"),
                        _T("InputLine"), ACRX_CMD_MODAL, inputLine);
```

4. Save, build, and load the *AU2014.arx* file into AutoCAD.
5. At the Command prompt, type **inputline** and press Enter.
6. In the drawing area, specify the start and endpoints for the line.
7. Unload the AU2014.arx file from AutoCAD.

## E8 Select Objects and Request a Keyword

This exercise explains how to use the *acedSSGet* and *acedGetKword* functions to select objects and request a color to apply to the selected objects.

1. In Microsoft Visual Studio, place the cursor after the line `#include "dbents.h"` in the source code file and press Enter twice.
2. Type the following code to define a function named *changeColor* which prompts the user for a keyword that determines the color that the selected objects should be assigned. You do not need to enter the lines of code that start with `//`.

```
static void changeColor()
{
    ads_name sset, ename;

    // Prompt the user for objects to modify
    if (acedSSGet(NULL, NULL, NULL, NULL, sset) == RTNORM)
    {
        long lSSCnt = 0;
        acedSSLength(sset, &lSSCnt);
```

```
        // Display the number of objects selected
        acutPrintf(_T("\nObjects selected: %i"), lSSCnt);

        TCHAR kWord[30] = _T("");
        TCHAR kDef[30] = _T("Red");

        // Prompt the user for a keyword/option
        acedInitGet(0, _T("1 2 3 Red Yellow Green Bylayer") );
        int retVal = acedGetKword(
            _T("\nEnter a color [Red/Yellow/Green/Bylayer] <Red>: "),
            kWord);

        // User entered a keyword or pressed enter
        if (retVal == RTNORM || retVal == RTNONE)
        {
            // Set the value that should be current
            // if the user presses Enter
            if (retVal == RTNONE)
            {
                _tcscpy(kWord, kDef);
            }

            // Loop through all the objects
            for (int loopCnt = 0; loopCnt < lSSCnt; loopCnt++)
            {
                // Get the next object from the selection set
                acedSSName(sset, loopCnt, ename);

                // Get the object id for the object
                // from the selection set
                AcDbObjectId objId;
                acdbGetObjectId(objId, ename);

                // Open the object for write
                AcDbEntity *pEnt;
                acdbOpenObject(pEnt, objId, AcDb::kForWrite);

                // Change the object's color based on
                // the keyword entered
                AcCmColor color;
                color.setColorMethod(AcCmEntityColor::kByACI);

                // Determine which color to assign to the object
                if (_tcscmp(kWord, _T("1")) == 0 ||
                    _tcscmp(kWord, _T("Red")) == 0)
                {
                    color.setColorIndex(1);
                } else if (_tcscmp(kWord, _T("2")) == 0 ||
                    _tcscmp(kWord, _T("Yellow")) == 0) {
                    color.setColorIndex(2);
                } else if (_tcscmp(kWord, _T("3")) == 0 ||
                    _tcscmp(kWord, _T("Green")) == 0) {
```

```
                    color.setColorIndex(3);
                } else if (_tcscmp(kWord, _T("Bylayer")) == 0) {
                    color.setColorMethod(AcCmEntityColor::kByLayer);
                }

                // Set the color for the object and then close it
                pEnt->setColor(color);
                pEnt->close();
            }
        }
    }
}
```

3. In the *acrxEntryPoint* function, make the following addition in bold.

```
acedRegCmds->addCommand(_T("AUCommands"), _T("InputLine"),
                        _T("InputLine"), ACRX_CMD_MODAL, inputLine);
acedRegCmds->addCommand(_T("AUCommands"), _T("ChangeColor"),
                        _T("ChangeColor"),
                        ACRX_CMD_MODAL | ACRX_CMD_USEPICKSET, changeColor);
```

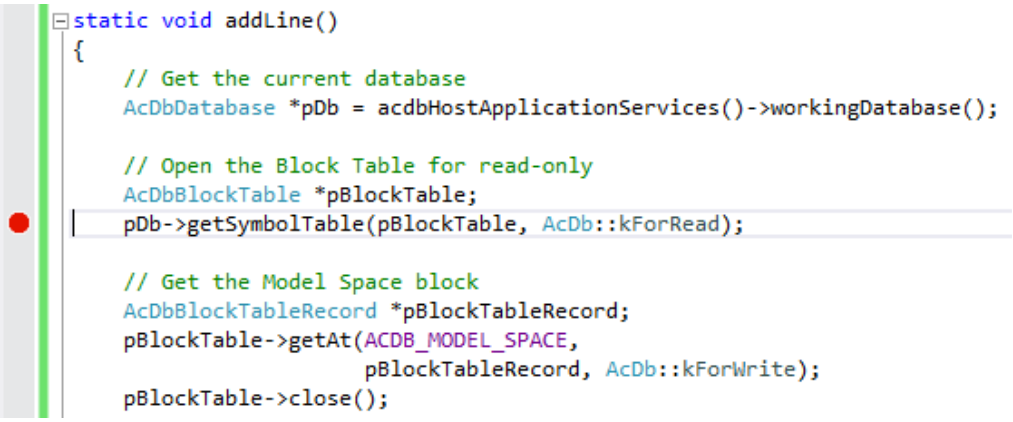
4. Save, build, and load the *AU2014.arx* file into AutoCAD.
5. At the Command prompt, type **changecolor** and press Enter.
6. In the drawing area, select objects in the drawing and press Enter.
7. At the *Enter a color [Red/Yellow/Green/Bylayer] <Red>:* prompt, type **r** and press Enter.
8. In the drawing area, select objects.
9. At the Command prompt, type **changecolor** and press Enter.
10. At the *Enter a color [Red/Yellow/Green/Bylayer] <Red>:* prompt, type **g** and press Enter.
11. Unload the AU2014.arx file from AutoCAD.

## E9 Debug an ObjectARX Application

This exercise explains how to build a project for debugging and debug the project in Microsoft Visual Studio.

1. In Microsoft Visual Studio, on the menu bar, click Build ➢ Configuration Manager.
2. In the Configuration Manager dialog box, Active Solution Configuration drop-down list, choose Debug. Click Close.
3. In the source code window, click in the leftmost margin across from the code line `pDb->getSymbolTable(pBlockTable, AcDb::kForRead);` in the *addLine* function to set a breakpoint.

The breakpoint is where the program will stop and wait for you to step through each line of code to be executed.



```
static void addLine()
{
    // Get the current database
    AcDbDatabase *pDb = acdbHostApplicationServices()->workingDatabase();

    // Open the Block Table for read-only
    AcDbBlockTable *pBlockTable;
    pDb->getSymbolTable(pBlockTable, AcDb::kForRead);

    // Get the Model Space block
    AcDbBlockTableRecord *pBlockTableRecord;
    pBlockTable->getAt(ACDB_MODEL_SPACE,
                       pBlockTableRecord, AcDb::kForWrite);
    pBlockTable->close();
```

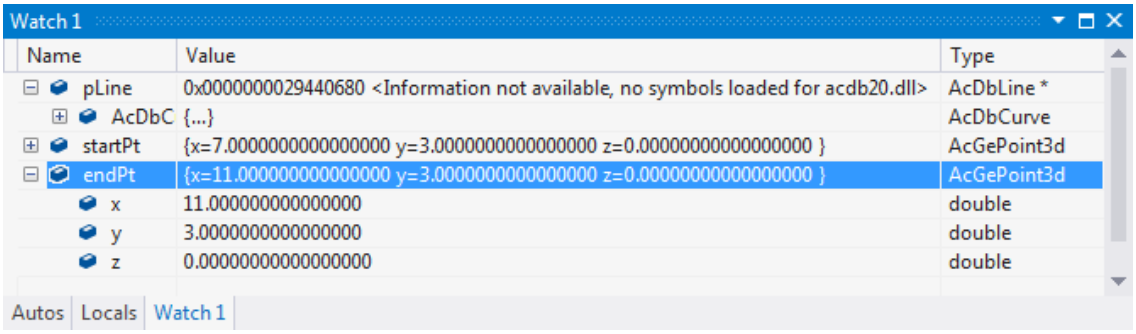
4. On the menu bar, click Debug ➢ Start Debugging.
5. If a Microsoft Visual Studio message box is displayed, and states that This Project is Out of Date, click Yes to build the project.

   The message is related to adding the breakpoint to the project. After clicking Yes, Visual Studio starts up AutoCAD 2015.
6. In the new AutoCAD 2015 session, create a new drawing and then display the Load/Unload Applications dialog box.
7. In the Load/Unload Applications dialog box, and browse to the location of the compiled debug version of the ARX file. Select *AU2014.arx* and click Load.
8. In the File Loading - Security Concern message box, click Load.
9. Click Close.
10. At the Command prompt, type **addline** and press Enter.

    Focus will shift to Microsoft Visual Studio and the line with the breakpoint will have a yellow arrow next to it, indicating it is waiting for you to start debugging.
11. Press F10 to begin stepping through each line of code and until the yellow arrow is to the left of the code line `pBlockTableRecord->appendAcDbEntity(pLine);`.
12. Hover the cursor over the `endPt` variable in the line that begins with `AcDbLine *pLine` to see its current value.

```
// Define the points that will be used to define the Line object
AcGePoint3d startPt(7.0, 3.0, 0.0);
AcGePoint3d endPt(11.0, 3.0, 0.0);

// Create the new Line object in memory
AcDbLine *pLine = new AcDbLine(startPt, endPt);
                                    endPt {x=11.000000000000000 y=3.0000000000000000 z=0.0000000000000000 }
// Add the new Line object to Model space
pBlockTableRecord->appendAcDbEntity(pLine);

// Close the Model space block
pBlockTableRecord->close();
```

You can also use the Watch window, which displays automatically while debugging a project.




13. Click Debug ➢ Continue to resume normal execution.

    You are returned to AutoCAD.

14. Switch back to Microsoft Visual Studio and click Debug ➢ Stop Debugging to end the debugging session.

15. Unload the AU2014.arx file from AutoCAD.

## E10 Build an ObjectARX Application for Release

This exercise explains how to build a project for release on other workstations.

1. In Microsoft Visual Studio, on the menu bar, click Build ➢ Configuration Manager.

2. In the Configuration Manager dialog box, Active Solution Configuration drop-down list, choose Release. Click Close.

3. In Microsoft Visual Studio, on the menu bar, click Build ➢ Build Solution.

4. The location of the build output is displayed in the Output window.

```
1>  AU2014.vcxproj -> C:\Datasets\Wednesday\SD6174-L Sparring with Autodesk®
ObjectARX® - Round 2 Stepping into the Ring\AU2014\x64\Release\AU2014.arx
1>FinalizeBuildStatus:
1>  Deleting file "x64\Release\AU2014.unsuccessfulbuild".
1>  Touching "x64\Release\AU2014.lastbuildstate".
```

5. Load the file into AutoCAD.

6. Try some of the commands defined in the ARX file: UNO, ADDLINE, MAKELAYER, LISTOBJECTS, INPUTLINE, and CHANGECOLOR.

   The commands should execute just as they did before. However, the release version of the ARX file can be copied and executed on other workstations executing AutoCAD 2015.

7. Unload the AU2014.arx file from AutoCAD.

### Extra - E1 Work with System Variables and Use Commands

This exercise explains how to use the *acedSetVar*, *acedGetVar*, and *acedCommandS* methods to work with system variables and execute a command at the command line.

1. In Microsoft Visual Studio, place the cursor after the line `#include "dbents.h"` in the source code file and press Enter once.

2. Type the following code to use the functions in the *acedCmdNF.h*:

```
#include "acedCmdNF.h"
```

3. Type the following code to define a function named *commandAndSysVar* which allows you to work with commands and system variables in the current drawing. You do not need to enter the lines of code that start with `//`.

```
static void commandAndSysVar()
{
    // Create a variable of the result buffer type
    struct resbuf rb, rb1;

    // Get the current value of the CIRCLERAD system variable
    acedGetVar(_T("CIRCLERAD"), &rb);
    acutPrintf(_T("\nCurrent CIRCLERAD: %.2f\n"), rb.resval);

    // Define the center point for the circle
    ads_point pt;
    pt[X] = 2.5; pt[Y] = 3.75; pt[Z] = 0.0;

    // Define the radius of the circle
    double rrad = 2.75;

    // Provide all values to the CIRCLE command
    if (acedCommandS(RTSTR, _T("._CIRCLE"), RTPOINT, pt,
                     RTREAL, rrad, RTNONE) != RTNORM)
    {
        acutPrintf(_T("\nError: CIRCLE command failed."));
    }
```

```
    // Pause for the center point
    if (acedCommandS(RTSTR, _T("._CIRCLE"), RTSTR, PAUSE,
                     RTREAL, rrad, RTNONE) != RTNORM)
    {
        acutPrintf(_T("\nError: CIRCLE command failed."));
    }

    // Set the value of CIRCLERAD to the previous value
    rb1.restype = RTREAL;
    rb1.resval.rreal = rb.resval.rreal;
    acedSetVar(_T("CIRCLERAD"), &rb1);
}
```

4. In the *acrxEntryPoint* function, make the following addition in bold.

```
acedRegCmds->addCommand(_T("AUCommands"), _T("ChangeColor"),
                        _T("ChangeColor"),
                        ACRX_CMD_MODAL | ACRX_CMD_USEPICKSET, changeColor);
acedRegCmds->addCommand(_T("AUCommands"), _T("CommandAndSysVar"),
                        _T("CommandAndSysVar"), ACRX_CMD_MODAL,
                        commandAndSysVar);
```

5. Save, build, and load the *AU2014.arx* file into AutoCAD.
6. At the Command prompt, type **commandandsysvar** and press Enter.
7. In the drawing area, specify a point to draw the second circle.
8. Unload the AU2014.arx file from AutoCAD.

## Extra - E2 Work with Input and Single Line Text Objects

This exercise explains how to use the *acedGetInt*, *acedGetString*, and *acedGetPoint* methods to get input from the user and then create single line text objects with the values entered.

1. In Microsoft Visual Studio, place the cursor after the line `#include "acedCmdNF.h"` in the source code file and press Enter twice.
2. Type the following code to define a function named *userInfo* which prompts the user for input and then adds two single-line text objects to model space. You do not need to enter the lines of code that start with `//`.

```
static void userInfo()
{
    // Request the user's age
    int nAge = 0;
    acedGetInt(_T("\nEnter your age: "), &nAge);

    // Request the user's name
    TCHAR cName[30];
```

```
    acedGetString(NULL, _T("\nEnter your name: "), cName);

    // Specify the insertion point for the first single line text object
    ads_point pt;
    acedGetPoint(NULL, _T("\nSpecify insertion point: "), pt);

    // Convert the entered age from an Integer to a character array
    TCHAR cAge[33];
    _itot(nAge, cAge, 10);

    // Build the string for the first text object
    TCHAR cVal1[512];
    _tcscpy(cVal1, _T("Age: "));
    _tcscat(cVal1, cAge);

    // Build the string for the second text object
    TCHAR cVal2[512];
    _tcscpy(cVal2, _T("Name: "));
    _tcscat(cVal2, cName);

    // Get the current database
    AcDbDatabase *pDb = acdbHostApplicationServices()->workingDatabase();

    // Get the current space object
    AcDbBlockTableRecord *pBlockTableRecord;
    Acad::ErrorStatus es = acdbOpenObject(pBlockTableRecord,
                           pDb->currentSpaceId(), AcDb::kForWrite);

    if (es == Acad::eOk)
    {
        AcGePoint3d ptIns(pt[0], pt[1], pt[2]);

        // Create the first text object at a height of 3.5
        AcDbText *pText1 = new AcDbText(ptIns, cVal1);
        pText1->setHeight(3.5);

        // Define the insertion point for the second text object
        ptIns.y = ptIns.y - 5;

        // Create the second text object at a height of 3.5
        AcDbText *pText2 = new AcDbText(ptIns, cVal2);
        pText2->setHeight(3.5);

        // Create a new ObjectId for the new Text objects
        AcDbObjectId text1Id, text2Id;

        // Add the new Text objects to the current space
        pBlockTableRecord->appendAcDbEntity(text1Id, pText1);
        pBlockTableRecord->appendAcDbEntity(text2Id, pText2);

        // Close the current space block
        pBlockTableRecord->close();
```

```
        // Close the new text objects
        pText1->close();
        pText2->close();
    } else {
        acutPrintf(_T("\nERROR: Block could not be opened for write."));
    }
}
```

3. In the *acrxEntryPoint* function, make the following addition in bold.

```
acedRegCmds->addCommand(_T("AUCommands"), _T("ChangeColor"),
                        _T("ChangeColor"),
                        ACRX_CMD_MODAL | ACRX_CMD_USEPICKSET, changeColor);
acedRegCmds->addCommand(_T("AUCommands"), _T("UserInfo "),
                        _T("UserInfo"), ACRX_CMD_MODAL, userInfo);
```

4. Save, build, and load the *AU2014.arx* file into AutoCAD.

5. At the Command prompt, type **userinfo** and press Enter.

6. Specify an integer value, then your first name, and an insertion point for the text objects to be drawn.

   Two single line text objects are added near the point specified in the drawing.

7. Unload the AU2014.arx file from AutoCAD.

## Extra - E3 Defining AutoLISP Functions

This exercise explains how to implement four custom AutoLISP functions: DTR, RTD, TAN, and EX_ALERT. These functions can be accessed from an AutoLISP program or the Command prompt when the AU2014.arx file is loaded.

1. In Microsoft Visual Studio, place the cursor after the line `#include "acedCmdNF.h"` in the source code file and press Enter twice.

2. Type the following code to define the callback functions and code used to setup AutoLISP functions. You do not need to enter the lines of code that start with `//`.

```
// Constant value of PI
#define PI 3.14159

/* Utility definition used to get an  array's  element  count. */
#define ELEMENTS(array) (sizeof(array)/sizeof((array)[0]))

/* Structure table used for AutoLISP functions:
   A string representing the AutoLISP function name,
   and a pointer to the callback function which must
   return an integer (int). */
```

```
struct func_entry { ACHAR *func_name; int (*func) (struct resbuf *); };

// Advanced Messaging
int Ex_Alert   (struct resbuf *rb);

// Math functions
int Dtr        (struct resbuf *rb);
int Rtd        (struct resbuf *rb);
int tangent    (struct resbuf *rb);

// Add the AutoLISP functions to the function table
static struct func_entry func_table[] = { {_T("Ex_Alert"), Ex_Alert},
                                          {_T("Dtr"), Dtr},
                                          {_T("Rtd"), Rtd},
                                          {_T("Tan"), tangent}
                                        };

/*-------------------------------------------------------------------------
   FUNCLOAD  --  Define this application's external functions.  Return
                 RTERROR on error, else RTNORM.                          */
static int funcload()
{
    int i;

    for (i = 0; i < ELEMENTS(func_table); i++) {
        if (!acedDefun(func_table[i].func_name, (short)i))
            return RTERROR;
    }
    return RTNORM;
}

/*-------------------------------------------------------------------------
   DOFUN -- Execute external function (called upon an RQSUBR request).
            Return value from the function executed, RTNORM or RTERROR. */
static int dofun()
{
    struct resbuf *rb;
    int val;

    if ((val = acedGetFunCode()) < 0 || val >= ELEMENTS(func_table)) {
        acdbFail(_T("Received nonexistent function code."));
        return RTERROR;
    }

    /* Get the arguments passed into the function */
    rb = acedGetArgs();

    /* Execute the callback function and return its status. */
    val = (*func_table[val].func)(rb);
    acutRelRb(rb);
    return val;
}
```

```
// Tangent - Tan function doesn't exist in AutoLISP by default
static int tangent(struct resbuf *rb)
{
    double dDistance;

    if (rb == NULL)
    {
        acutPrintf(_T("Improper use of function:
                       (Tan <angle in radians>)"));
        return RTERROR;
    }

    if (rb->restype == RTREAL) {     /* Can accept an interger */
        dDistance = rb->resval.rreal;
    } else if (rb->restype == RTLONG) {
        dDistance = rb->resval.rlong;
    } else if (rb->restype == RTSHORT) {
        dDistance = rb->resval.rint;
    } else {
        acdbFail(_T("Argument one should be a number"));
        return RTERROR;
    }

    acedRetReal(tan(dDistance));

    return RTNORM;
}

/*-------------------------------------------------------------------------*/
/* This is the implementation of the actual external function */
static ads_real rdtr(ads_real deg)       /* Degrees to Radians */
{
       ads_real retVal;

    retVal = (deg * (PI / 180));  // PI is defined above
    return retVal;
}

/*-------------------------------------------------------------------------*/
/* DTR -- First set up the arguments, then call the RDTR function */
static int Dtr(struct resbuf *rb)
{
    double x;

    if (rb == NULL)
    {
        acutPrintf(_T("Improper use of function: (dtr <degrees>)\n"));
        return RTERROR;
    }

    if (rb->restype == RTSHORT) {     /* Can accept an interger */
```

```
        x = (ads_real) rb->resval.rint;
    } else if (rb->restype == RTREAL) {
        x = rb->resval.rreal;         /* or a real number */
    } else {
        acdbFail(_T("Argument should be an real or integer number."));
        return RTERROR;
    }

    acedRetReal(rdtr(x));            /* Call the function itself, and
                                        return the value to AutoLISP */
    return RTNORM;
}

/*-------------------------------------------------------------------------*/
/* This is the implementation of the actual external function */
static ads_real rrtd(ads_real rad)       /* Radians to Degrees */
{
    ads_real retVal;

    retVal = (180 * (rad / PI));  // PI is defined above
    return retVal;
}

/*-------------------------------------------------------------------------*/
/* RTD -- First set up the argument, then call the RRTD function */
static int Rtd(struct resbuf *rb)
{
    double x;

    if (rb == NULL)
    {
      acutPrintf(_T("Improper use of function: (rtd <radians>)\n"));
      return RTERROR;
    }

    if (rb->restype == RTSHORT) {     /* Can accept an interger */
        x = (ads_real) rb->resval.rint;
    } else if (rb->restype == RTREAL) {
        x = rb->resval.rreal;         /* or a real number */
    } else {
        acdbFail(_T("Argument should be an real or integer number."));
        return RTERROR;
    }

    acedRetReal(rrtd(x));            /* Call the function itself, and
                                        return the value to AutoLISP */
    return RTNORM;
}

/*-------------------------------------------------------------------------*/
/* EX_ALERT -- Extended Alert, Message, Button, ICON                       */
static int Ex_Alert(struct resbuf *rb)
```

```
{
    ACHAR * message = _T("");
    ACHAR * caption = _T("AutoCAD Message");
    int buttons = MB_OK;
    int icon = 0, defbutton = 0;

    int retVal;
    struct resbuf *rbList = NULL, *transList = NULL;

    if (rb == NULL)
    {
        acutPrintf(_T("Improper use of function:
                      (ex_alert <message> ([caption] ([BUTTON] [ICON]))"));
        return RTERROR;
    }

    if (rb->restype == RTSTR) {     /* Accept an string */
        message = rb->resval.rstring;
    } else {
        acdbFail(_T("Argument should be a string"));
        return RTERROR;
    }

    rb = rb->rbnext;

    if (rb != NULL)
    {
        if (rb->restype == RTSTR) {     /* Accept an string */
            caption = rb->resval.rstring;
        } else {
            acdbFail(_T("Argument should be a string"));
            return RTERROR;
        }

        rb = rb->rbnext;

        if (rb != NULL)
        {
            if (rb->restype == RTSHORT) {     /* Accept an integer */
                buttons = rb->resval.rint;
            } else {
                acdbFail(_T("Argument should be an integer"));
                return RTERROR;
            }

            rb = rb->rbnext;

            if (rb != NULL)
            {
                if (rb->restype == RTSHORT) {     /* Accept an integer */
                    icon = rb->resval.rint;
```

```
                    switch (icon)
                    {
                        case 10:
                            icon = MB_ICONSTOP;
                            break;
                        case 20:
                            icon = MB_ICONQUESTION;
                            break;
                        case 30:
                            icon = MB_ICONEXCLAMATION;
                            break;
                        case 40:
                            icon = MB_ICONINFORMATION;
                            break;
                        default:
                            icon = 0;
                    }
                } else {
                    acdbFail(_T("Argument should be an integer"));
                    return RTERROR;
                }
            }
        }
    }

    retVal = MessageBox(adsw_acadMainWnd(),message,caption,buttons + icon);

    acedRetInt(retVal);            /* Call the function itself, and
                                      return the value to AutoLISP */
    message = NULL;
    delete message;

    caption = NULL;
    delete caption;

    return RTNORM;
}
```

3. In the *acrxEntryPoint* function, make the following additions in bold. These additions define user-defined variables that can be accessed from AutoLISP or using the ! (exclamation point) at the Command prompt.

```
acedRegCmds->addCommand(_T("AUCommands"), _T("UserInfo "),
                        _T("UserInfo"), ACRX_CMD_MODAL, userInfo);

int retVal;
struct resbuf *symVal;

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 0;
retVal = acedPutSym(_T("MB_OK"), symVal);
```

```
symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 1;
retVal = acedPutSym(_T("MB_OKCANCEL"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 2;
retVal = acedPutSym(_T("MB_ABORTRETRYIGNORE"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 3;
retVal = acedPutSym(_T("MB_YESNOCANCEL"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 4;
retVal = acedPutSym(_T("MB_YESNO"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 5;
retVal = acedPutSym(_T("MB_RETRYCANCEL"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 16384;
retVal = acedPutSym(_T("MB_HELP"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 0;
retVal = acedPutSym(_T("MB_DEFBUTTON1"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 256;
retVal = acedPutSym(_T("MB_DEFBUTTON2"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 512;
retVal = acedPutSym(_T("MB_DEFBUTTON3"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 768;
retVal = acedPutSym(_T("MB_DEFBUTTON4"), symVal);


symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 0;
retVal = acedPutSym(_T("MB_ICONNONE"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 20;
retVal = acedPutSym(_T("MB_ICONQUESTION"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 40;
retVal = acedPutSym(_T("MB_ICONINFORMATION"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 10;
```

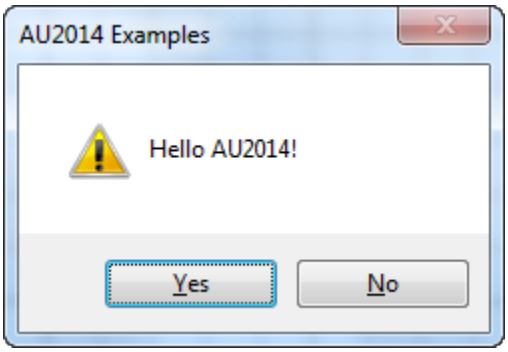
```
retVal = acedPutSym(_T("MB_ICONSTOP"), symVal);

symVal = acutNewRb(RTSHORT);
symVal->resval.rint = 30;
retVal = acedPutSym(_T("MB_ICONEXCLAMATION"), symVal);

symVal = NULL;
delete symVal;
```

4. Save, build, and load the *AU2014.arx* file into AutoCAD.
5. At the Command prompt, type **(dtr 90)** and press Enter.

   The function returns a value of 1.57079.
6. Type **(rtd PI)** and press Enter.

   The function returns a value of 180.0.
7. Type **(ex_alert "Hello AU2014!" "AU2014 Examples" MB_YESNO MB_ICONEXCLAMATION)** and press Enter.

   The following message box is displayed:




8. Click Yes or No.

   If you click Yes, a value of 6 is returned. Clicking No returns a value of 7.
9. Unload the AU2014.arx file from AutoCAD.

# Appendixes

This section contains the appendixes that explain how to download and install the ObjectARX SDK and Wizard.

## A1 Download and Install the ObjectARX 2015 SDK

This appendix explains how to install the ObjectARX SDK which is required for you to compile an ObjectARX program for use in AutoCAD.

1. In your Web browser, navigate to the URL *http://autodesk.com/developautocad* and click the ObjectARX link.
2. On the ObjectARX web page, click the License & Download link.
3. Fill out the form located near the bottom of the page, and select the appropriate ObjectARX SDK option based on the release you are working with. Click Submit.
4. If you need to target multiple releases, download the oldest ObjectARX release. So if you need to target AutoCAD 2013 and 2014, select the ObjectARX for AutoCAD 2014 option.
5. In the File Download dialog box, click Save and browse to the location you want to download the ObjectARX SDK file to. Click Save to continue downloading the file.
6. Click Run to begin extracting the files contained in the ObjectARX SDK. Click Run again.
7. In the ObjectARX 2015 install window, click Install. Keep the default install folder.

## A2 Download and Install the ObjectARX Wizard

This appendix explains how to install the ObjectARX Wizard.

1. In your Web browser, navigate to the URL http://autodesk.com/developautocad and click the ObjectARX 2015 Wizard link under the Samples and Documentation section.
2. In the File Download dialog box, click Save and browse to the location you want to download the ObjectARX Wizard file to. Click Save to continue downloading the file.
3. Click Open Folder to browse to the downloaded ZIP file.
4. Extract the contents of the ZIP file and double-click ObjectARXWizards.msi.
5. In the ObjectARX 2015 Wizards installer, click Next.
6. Optionally, enter a RDS symbol you want to use with the wizard.
7. Specify the location where the ObjectARX SDK is located and the location of AutoCAD. Click Next.

8. Click Install Now.

9. Click Close after installation has completed.

## A3 Working with VS Express 2012 for Desktop

This appendix explains the differences between Microsoft Visual Studio 2012 and VS Express 2012 for Desktop.

VS Express 2012 for Desktop can be downloaded from *http://www.microsoft.com/en-us/download/details.aspx?id=34673*. Be sure to install Update 4 as well for Visual Studio 2012.

For the most part VS 2012 and VS Express are nearly identical with main two exceptions; VS Express doesn't support ATL or MFC.

- ATL (Active Template Library) is used to create custom COM (Component Object Model) objects.
- MFC (Microsoft Foundation Class library) is used to develop custom dialog boxes.

To create a new project for ObjectARX 2015 in VS Express 2012 for Desktop, you follow the steps outlined in E1 - Create a New Project from Scratch (Microsoft Visual Studio 2012) and make a change to the Ignore Specific Default Libraries property.

The following explains how to make the proper changes to the Ignore Specific Default Libraries property:

1. In the Solution Explorer, select AU2014. Right-click and choose Properties.

2. In the Property Pages dialog box, expand Configuration Properties.

3. Under Configuration Properties, expand Linker and select Input.

    a. From the Configuration drop-down list, select All Configurations.

    b. Click in the Ignore Specific Default Libraries field.

    c. Click the drop-down arrow, and choose <Edit…>.

    d. In the Ignore Specific Default Libraries dialog box, type **atls.lib** and click OK.

4. Click OK to save the change to the project's properties.

The project shouldn't fail to build because of a linker issue related to ATL.