SD6148 - Sparring with Autodesk® ObjectARX®—Round 1 Stepping into the Ring

Lee Ambrosius – Autodesk, Inc.

Principal Learning Content Developer – IPG

AutoCAD Products – Learning Experience

Where Am I and Who Should Be Here

You are in session:
SD6148 - Sparring with Autodesk® ObjectARX®—Round 1
Stepping into the Ring

You should know:

- AutoCAD 2015 (or AutoCAD 2013 and later)
- Previous programming experience, C++ would be a plus (no pun intended)

Overview

AutoCAD can be extended through a number of different programming options, such as:

- AutoLISP
- Managed .NET (VB.NET, C#)
- ActiveX/COM enabled programing languages

This session covers:

- Setting up an ObjectARX project
- Basics of ObjectARX

Who Am I?

My name is Lee Ambrosius

- Principal Learning Content Developer at Autodesk
- AutoCAD user for over 20 years
- Work on the Customization, Developer, and CAD Administration documentation
  - Customizing and programming for over 18 years
  - AutoLISP/DCL, VBA, Managed .NET, and ObjectARX/MFC
- Author and Technical Editor
  - AutoCAD and AutoCAD LT Bible Series
  - AutoCAD Customization Platform Series

Getting Started

Session Handouts

The handouts are broken into two parts:

- Supplemental – Content covered in the lecture and for the flight back
- Exercises – What will be covered in the lab

What You Will Learn Today

By the end of the lecture, you will know how to:

- What is needed to get started with ObjectARX
- Create a command and work with objects in a drawing
- Display messages to the user and request user input
- Load an ObjectARX application into AutoCAD

What You Need Before Getting Started

The following will be used in the lecture and lab:

- AutoCAD 2015 and ObjectARX 2015 SDK
- Microsoft Visual Studio 2012

When you get back to work, you will want to utilize the following:

- ObjectARX Reference Guide
- ObjectARX Developer's Guides

What is ObjectARX?

ObjectARX is a software development library or kit.

- Allows you to communicate directly with AutoCAD
- Not a programming language like AutoLISP or VB.NET
- Requires you to know and use
  - C++ and Microsoft Visual Studio on Windows
  - Objective-C and Xcode on Mac OS X

Need to Know About C++ Before Starting

C++ and Objective-C are case sensitive languages

- ACDBLINE and acdbline is not the same as AcDbLine
- Most statements can span multiple lines
- Visual Studio will let you know when something has been typed in correctly

Need to Know About C++ Before Starting

Pointers

- Don’t actually hold a value
- Refers to a location in memory; memory address
- Often used to refer to other variables
- * is used when declaring a variable as a pointer

```
int *nCnt = 0;
```

Need to Know About C++ Before Starting

When a new object is created, a variable is defined as a pointer.

The pointer is then passed to AutoCAD so it has direct access to the object's location in memory.

```
AcDbLine *pLine = new AcDbLine(startPt, endPt);
```

Need to Know About C++ Before Starting

References

- Don’t actually hold a value
- Refers to the value of another variable; think of an alias
- & is used when declaring a variable as a reference
- Used to return more than one value from a function

Need to Know About C++ Before Starting

The following is an example of referencing:

```
long lSSCnt = 0;

acedSSLength(sset, &lSSCnt);
```

The address of the *lSSCnt* variable is passed to the *acedSSLength* function.

The *lSSCnt* variable now contains an integer that contains the number objects in the selection set.

Common C++ Syntax

//	Denotes a statement that should not be executed. // Creates new line
/* … */	Denotes a series of statements that should not be executed. /* Defines a new command. The new command creates a circle and a line. */
;	Denotes the end of a statement. int iCnt = 0;
#include	Imports classes and functions for use in the current code module. #include "arxHeaders.h"

Common C++ Syntax

retType funcName (*vars* …)
{
…
}
	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.
	
	int addValues (int val1, int val2)
	{
	return val1 + val2;
	}

ObjectARX SDK Files

Access ObjectARX Libraries

ObjectARX is made up of a number of library and header files.

- Libraries provide access to various features available in the AutoCAD program.
- Header files provide definitions of the classes and functions that are supported with ObjectARX.

The libraries and headers are installed as part of the ObjectARX SDK. By default, the latest release is installed to:

- C:\ObjectARX 2015

Access ObjectARX Libraries

The libraries are broken up into shared, and 32-bit and 64-bit specific files:

- inc
- inc-win32 and lib-win32
- inc-x64 and lib-x64

Common ObjectARX Libraries

Most ObjectARX classes start with a four letter prefix to help identify what it is used for.

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.

Snipet of dbents.h (AcDbLine class)

```
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&);
```

Defining a New Command

Defining a New Command

The *acedRegCmds* macro is used to register a new command.

Commands require:

- Command group name
- Global or international command name
- Local command name
- Command flag(s)
- Name of the function to execute

Defining a New Command

The following example registers a command named Hello:

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

List of the commonly used Command Flags:

ACRX_CMD_TRANSPARENT or *ACRX_CMD_MODAL*	Defines a command as transparent or modal
ACRX_CMD_USEPICKSET	Pick first selection is allowed
ACRX_CMD_NOPAPERSPACE	Command is not allowed in paper space

Accessing AutoCAD and Drawing Objects

Accessing AutoCAD and Drawing Objects

Before you can working with drawing objects, you must obtain a database from the:

- Current drawing
- Document Manager

The following gets a pointer to the database of the current drawing:

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

Accessing AutoCAD and Drawing Objects

The following gets a pointer to the database of the current drawing from the DocumentManager:

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

Accessing AutoCAD and Drawing Objects

With the current database you can:

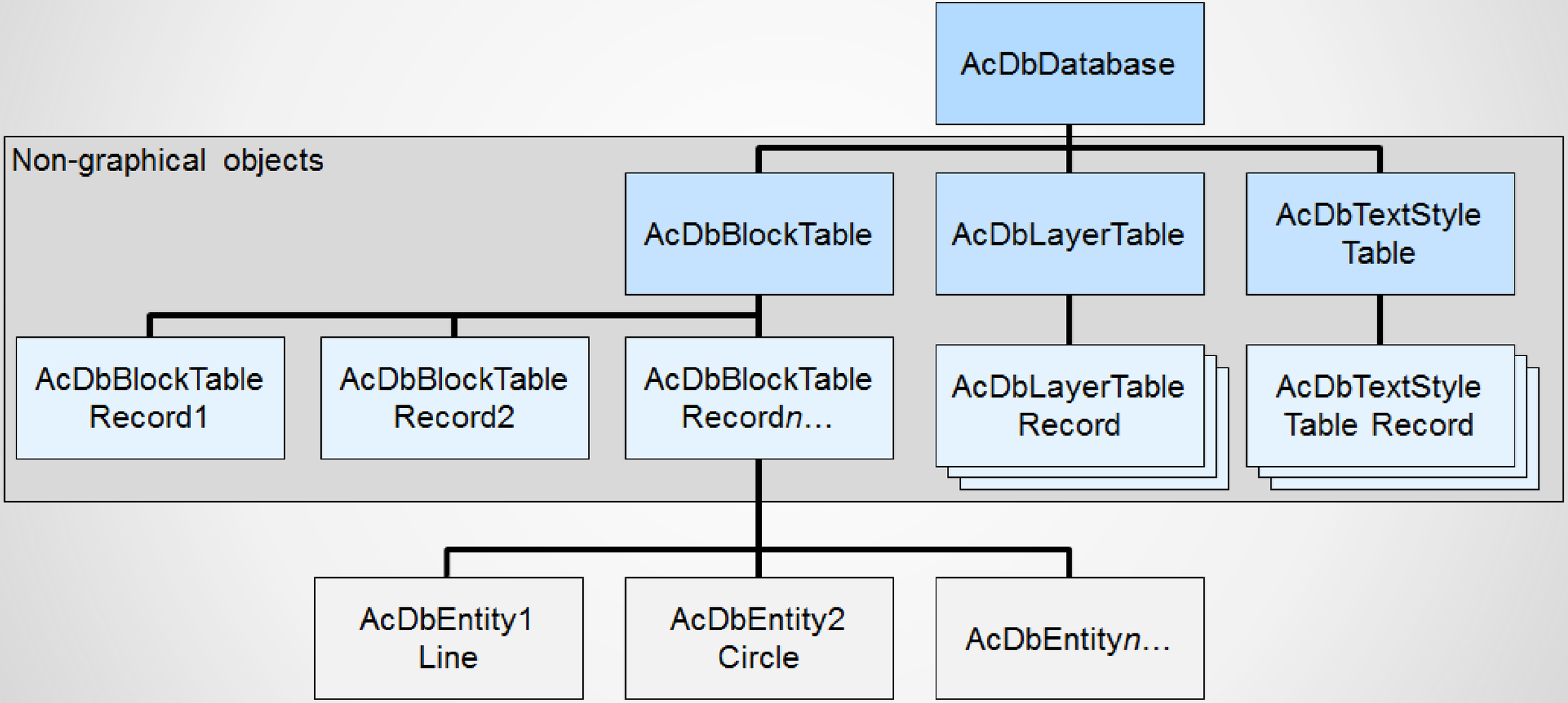
- Query the contents of a drawing
- Add and modify non-graphical and graphical objects
- Change the values of system variables saved to the drawing

Accessing AutoCAD and Drawing Objects

A drawing can contain:

- Nongraphical objects; symbol tables and dictionaries
- Graphical objects

Database Hierarchy



Non-Graphical Objects

Non-graphical objects are:

- Text, dimension, multileader, and table styles
- Groups
- Blocks
- Named viewports, views, and UCSs
- Layouts
- Among others…

Graphical Objects

Objects you add and interact with in the drawing area:

- Lines
- Circles
- Polylines
- Helixes
- 3D solids
- Among others…

Graphical Objects

Are added to:

- Model space (*MODEL_SPACE)
- Paper space (*PAPER_SPACE0, *PAPER_SPACE1, …)
- Block definition

Model space and paper space are special named block definitions.

Opening and Closing Objects

Existing objects need to be opened before you can:

- Read its properties
- Modify it; write to the object
- Add or append a new object

The *acdbOpenObject* and *getAt* methods are used to open an object for read or write, among a few others.

Opening and Closing Objects

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

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

Opening and Closing Objects

The read/write state of an object can be changed by using:

- *upgradeOpen* method – Read to write
- *downgradeOpen* method – Write to read

Opening and Closing Objects

An object that was opened, must be closed.

The *close* method closes an opened object.

```
// Close the block table
pBlockTable->close();

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

Creating a New Object

The *new* keyword is used to create a new object in memory.

```
AcDbLine *pLine = new AcDbLine(startPt, endPt);
```

The *new* keyword is followed by a data type, such as int or AcDbLine.

The constructor of the data type is called, and some data types support multiple constructors.

```
AcDbLine();
AcDbLine(const AcGePoint3d& start, const AcGePoint3d& end);
```

Adding New Objects

After a new object has been created, it can be appended to a drawing.

When you want to add a new object, you commonly use:

- *add* method – Adding a new table entry
- *appendAcDbEntity* method – Adding a new object to a block, such as model space or block table record

Parent object must be open for write before adding or appending a new object.

Opening and Closing Objects

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

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

Working with System Variables

Working with System Variables

The value of a system variable can be written or read using:

- *acedSetVar* method – Writes a value
- *acedGetVar* method – Reads a value

Values are structured as a result buffer (resbuf)

The value of a resbuf are control by two properties:

- *restype* property – Data type
- *resval* property – Value to assign or return

Working with System Variables

```
// 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);
```

Executing Commands

Executing Commands

Commands can be executed using:

- *acedCommandS* method – Executes a command and options based a supplied list; data type, value pairings
- *acedCommandC* method – Executes a command and options, but utilizes a callback function

Executing Commands

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

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

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

Executing Commands

Command can be paused for input with the PAUSE value.

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

Executing Commands

Commands can also be executed using the *sendStringToExecute* method.

The *sendStringToExecute* method accepts a string.

```
// 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);
```

Invoking AutoLISP Expressions

Invoking AutoLISP Expressions

AutoLISP expressions defined with the c: prefix can be invoked.

Returned value can be accessed.

Use the *acedInvoke* method to evaluate an AutoLISP expression.

Invoking AutoLISP Expressions

```
// (defun c:myadd ( num1 num2 / )(+ num1 num2))
static void executeLSP()
{
    struct resbuf *pLISPArgs;
    pLISPArgs = acutBuildList(RTSTR, _T("c:myadd"), RTLONG, 2, RTREAL, 0.75, RTNONE);
    struct resbuf *pResVal = NULL;
    int nRetVal = acedInvoke(pLISPArgs, &pResVal);

    if (nRetVal != RTERROR)
    {
        acutPrintf(_T("\nValue: %.2f\n"), pResVal->resval.rreal);

        acutRelRb(pLISPArgs);
        acutRelRb(pResVal);
    } else {
        acutPrintf(_T("\nERROR: AutoLISP expression couldn't be invoked."));
    }
}
```

Requesting User Input

Requesting User Input

An application can prompt for a :

- Point
- Number (integer or real)
- Distance or angle
- Object
- String and/or keyword

Requesting User Input

Dialog boxes can be implemented using :

- MFC on Windows
- Cocoa/Qt on Mac OS X

Requesting User Input

Most user input functions begin with *acedGet*:

- *acedGetInt* – Prompts for an integer
- *acedGetReal* – Prompts for a real or double value
- *acedGetString* – Prompts for a string
- *acedGetPoint* – Prompts for a point
- *acedGetKword* – Prompts for a keyword

Requesting User Input

```
// Prompts for an integer
int nAge = 0;
acedGetInt(_T("\nEnter your age: "), &nAge);

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

Selecting Objects

acedInitGet allows you to customize the behavior of the next *acedGetxxx* method.

The user input functions that begin with *acedGet* do not allow you to select objects.

The following functions are used to select objects:

- *acedEntsel* – Select a single object
- *acedSSGet* – Invokes the standard Select Objects prompt

Selecting Objects

```
// Prompts for a single graphical object
ads_point ePt;
ads_name eName;
acedEntSel(_T("\nSelect an entity: "), eName, ePt)

// Prompts for a selection set
ads_name sset;
acedSSGet(NULL, NULL, NULL, NULL, sset);
```

Selecting Objects

```
// Returns the first object in a selection set
ads_name eName;
acedSSName(sset, 0, eName);

// Gets the number of entities in a selection set
long lSSCnt = 0;
acedSSLength(sset, &lSSCnt);

// Release a selection set
acedSSFree(sset);
```

Return Values from Requesting User Input

User input functions return a value which can be used to determine how the user responded.

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

Providing Feedback to the User

In addition to getting input from a user, you can provide messages to users using:

- *acutPrintf* – Displays a message at the command prompt
- *acedAlert* – Displays a basic message box
- *acedTextScr* – Displays the Text window
- *acedGraphScr* – Switches focus to the drawing area

Final Thoughts and Questions

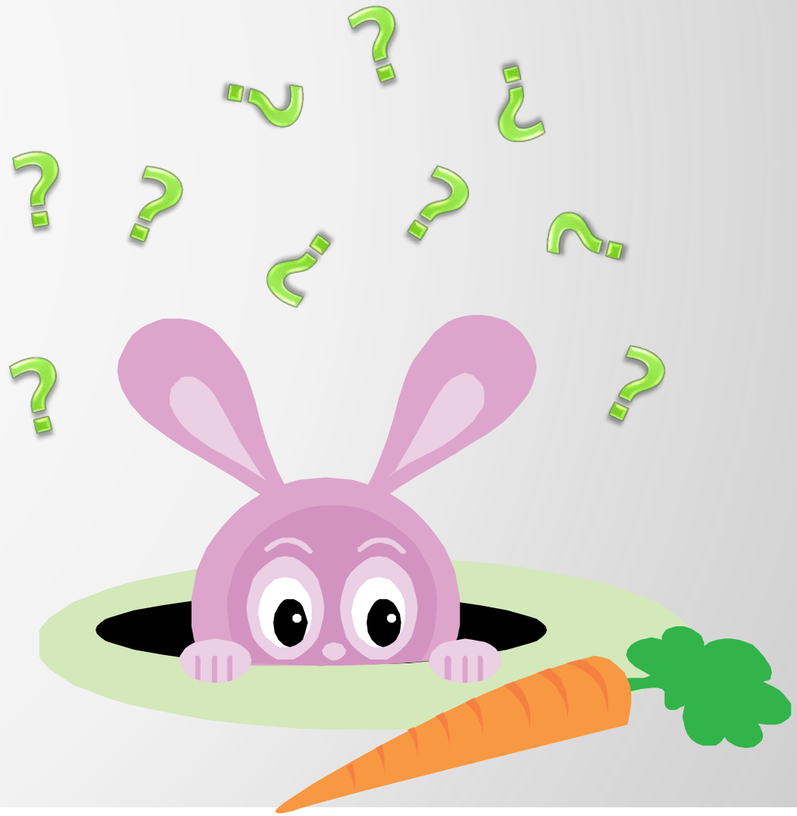
Final Thoughts and Questions

Scripting and programming can

- enhance productivity
- improve or introduce new workflows

Programming has many similarities to the rabbit hole in *Alice's Adventures in Wonderland* by Lewis Caroll. Both

- are virtually endless, and
- hold many mysteries that are waiting to be discovered



Another Related Session

Check out the handouts for the following session for more information on working with ObjectARX:

SD6174-L - Sparring with Autodesk® ObjectARX®—Round 2 Stepping into the Ring
Presented by: Lee Ambrosius

SD4861 - Creating AutoCAD Cross-Platform Plug-ins
Presented by: Fernando Malard

Closing Remarks

Thanks for choosing this session and hope you got something out of it.

Do not forget to complete the online evaluation.

If you have any further questions contact me via:

email: lee.ambrosius@autodesk.com
twitter: http://twitter.com/leeambrosius

Enjoy the rest of the conference.

Autodesk is a registered trademark of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries. All other brand names, product names, or trademarks belong to their respective holders. Autodesk reserves the right to alter product and services offerings, and specifications and pricing at any time without notice, and is not responsible for typographical or graphical errors that may appear in this document. © 2013 Autodesk, Inc. All rights reserved.