

ETRAGE Automation COM Interface (ACI) for Pro/ENGINEER Tutorial

ETRAGE Automation COM Interface (ACI) for Pro/ENGINEER Tutorial	1
Introduction	1
In-process vs. Out-of-process ACI	4
COM Server's Object Model.....	2
On Interfaces and Classes	2
Run-Time... ..	2
Step-by-step in VB.NET	3
Preparation of VB.NET Project	3
Connecting to ACI – IServer	3
Starting the Pro/ENGINEER – IProE	3
Loading the Part – IPart.....	3
Highlighting the Feature – IFeature	4
Disconnecting from ACI	4
Summary.....	4
Conclusion	4

Introduction

COM is Microsoft's object-oriented interoperability technology that is language and location independent. COM objects can communicate with any client language that conforms to the COM specification¹ and can be executed remotely over the network if so desired. "COM server" is executable (EXE or DLL) that implements set of COM objects.

Automation COM Interface (ACI) for Pro/ENGINEER, as its name suggests, is the **COM server that encapsulates PTC's Toolkit API's for Pro/ENGINEER**. It implements a set of COM objects that encapsulate internal Pro/ENGINEER objects (such as parts, features, parameters or dimensions), so operations that the client² invokes while working with these COM objects, are translated to native Pro/ENGINEER APIs (Pro/TOOLKIT).

This way, client is shielded from complexities of Pro/TOOLKIT and can use simple scripting languages for their automation needs (instead of just C/C++, as required by Pro/TOOLKIT).

ACI takes care of many technical quirks in Pro/TOOLKIT and exposes very user-friendly API (compared to Pro/ENGINEER Toolkit), so application developer can get right down to business very quickly, with smooth learning curve.

¹ This includes all .NET languages (such as VB.NET and C#) as well as VB6, C++, Tcl...

² That is, client program.

COM Server's Object Model

On Interfaces and Classes

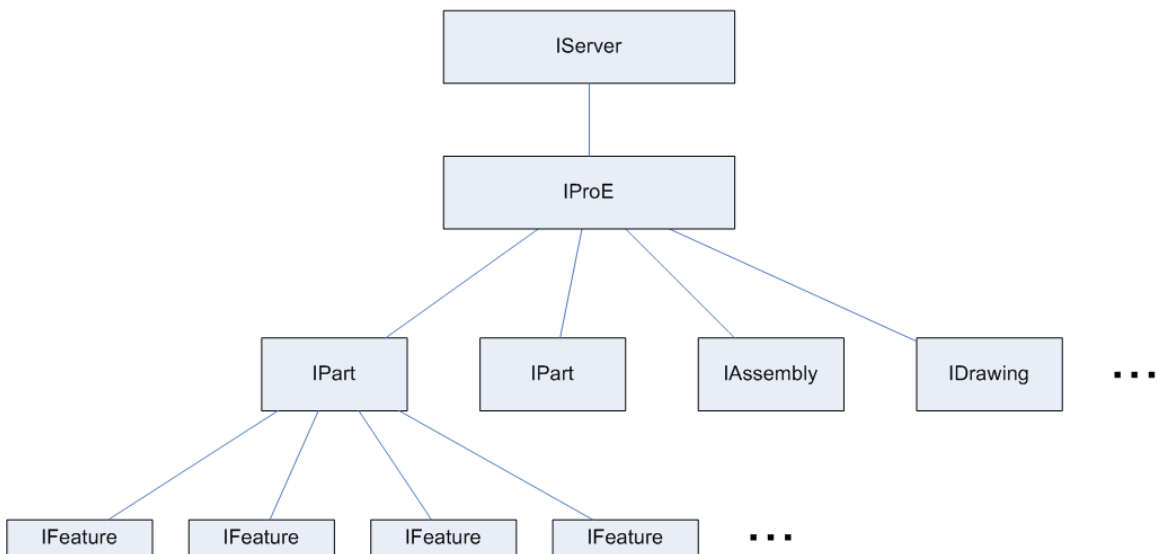
ACI for Pro/ENGINEER's API (or "object model", in COM terminology) consists of set of interfaces and classes. **Interface** is set of methods (a.k.a. functions, routines, operations...), and is what client actually uses. **Class** is physical implementation of an interface (or set of interfaces) and is not of concern to the client except in few special situations³.

COM supports multiple interfaces per class, but ACI for Pro/E deliberately implements exactly one (Pro/E related) interface per class, to avoid confusion that might arise otherwise. If class has name "SomeName", its interface's name will be always be in form "ISomeName".

Run-Time...

In run-time, class gets instantiated (and object of that class created), so client can call methods of interface(s) implemented by that class.

For example, if we have 2 parts, one assembly and one drawing loaded in Pro/ENGINEER session, run-time COM objects will form aggregation hierarchy similar to this:



This tutorial will illustrate how to programmatically build (part of) this hierarchy, and should give you a decent idea of the basic "philosophy" of how ACI for Pro/E is used.

³ The only situation in which the client would access a class directly is initial instantiation of Server class. All other classes are internally instantiated by methods in ACI's object model, so client never works with them directly, but through their interfaces.

Step-by-step in VB.NET

This tutorial assumes you have Visual Basic .NET 2003, but is compatible with other Visual Studio versions as well. If you use language other than VB.NET, you can still follow the tutorial and achieve good understanding of ACI basics – philosophy is the same in all client languages, only syntax will be different.

Preparation of VB.NET Project

Create a new Visual Basic project. You'll probably want a "Windows Application" project, so you can easily bind the code to event such as button click.

Before you can use ACI's object model, you must reference `ProeServer` type library in your VB.NET project. This is done from *Project / Add Reference / COM tab*.

Connecting to ACI – IServer

`IServer` is default interface of ACI's **root object**. Before client⁴ can do anything with ACI, it must instantiate `Server` class so it can get reference to `IServer` interface. In VB.NET, this is done by the following piece of code:

```
Dim server As New ProeServer.Server
```

This line of code will effectively load DLL (in case of in-process COM server) or start the EXE (for out-of-process COM server) that implements ACI.

`IServer` represents the COM Server itself. It contains information, such as location of debug log, which is not directly related to Pro/ENGINEER. Its main purpose, however, is to start Pro/ENGINEER and create object that represents it (`IProE`).

Starting the Pro/ENGINEER – IProE

Once you got hold of `IServer`, you can start the Pro/ENGINEER itself using one of `IServer`'s methods:

```
Dim proe As ProeServer.IProE = server.StartProE()
```

This will physically start latest version of Pro/ENGINEER installed on the machine that is currently running ACI. Returned interface allows the client to manipulate the started Pro/ENGINEER session.

Loading the Part – IPart

Following code loads the part from disk into Pro/ENGINEER session:

```
Dim part As ProeServer.IPart = proe.LoadPart("my_part.prt")
```

⁴ VB.NET program, in this case.

Once you loaded the model, you can show it in the Pro/ENGINEER window:

```
part.Visible = True
```

Highlighting the Feature – IFeature

Let's say we want to highlight the feature with given name. To do that, we need to get the reference to feature's COM interface...

```
Dim feature As ProeServer.IFeature = _  
    part.Features.ItemByName("EXTRUDE_1")
```

...then get associated "selection" object and highlight it...

```
feature.ToSelection().Highlight()
```

Disconnecting from ACI

ACI will automatically shut-down when your VB.NET application ends its execution. This happens because .NET run-time releases references to all COM objects previously held by the applications.

If you want to shut-down ACI (and any running Pro/ENGINEER with it) in the "middle" of your application's execution, you should do it like this:

```
' Release all references, so ACI can unload itself.  
server = Nothing  
proe = Nothing  
part = Nothing  
feature = Nothing  
  
' NOTE: VB .NET does not really release the reference to the COM  
' object until the reference is garbage collected.  
GC.Collect()  
GC.WaitForPendingFinalizers()
```

Summary

When we put these lines of code together, the whole program to load the part and highlight the feature looks like this:

```
Dim server As New ProeServer.Server  
Dim proe As ProeServer.IProE = server.StartProE()  
Dim part As ProeServer.IPart = proe.LoadPart("my_part.prt")  
part.Visible = True  
Dim feature As ProeServer.IFeature = _  
    part.Features.ItemByName("EXTRUDE_1")  
feature.ToSelection().Highlight()
```

Appendix: In-process vs. Out-of-process ACI

ACI comes in two variants: in-process and out-of-process. In-process ACI is executed within address space of client process and is physically implemented

as DLL. Out-of-process ACI executes as a separate process and is implemented as EXE.

Following table provides general guidelines the client should use when choosing between in-process and out-of-process ACI (or any other COM server):

In-process (DLL)	Out-of-process (EXE)
Better performance due minimal COM overhead – all COM calls are done locally within the same process.	Poorer performance due significant COM overhead – COM calls must be “marshaled” over the process and (possibly) machine boundaries.
Cannot execute remotely.	Can be executed remotely (over the LAN or even the Internet).
Error in ACI will probably bring the client down with it. The opposite also holds true: error in client can adversely affect ACI.	Bugs in ACI are gracefully handled in client. Opposite is also true: bug in client will not catastrophically crash ACI.
Limited security: ACI is executed under the user that started client process.	Security can be precisely configured through “dcomcnfg” tool. ACI can be executed as a user different from client and can be configured to allow access only to certain users.
Supports only one Pro/ENGINEER session per client (separate clients can still control separate Pro/ENGINEER sessions).	One client can access multiple Pro/ENGINEER sessions concurrently.

NOTE: There is also a third flavor of the ACI called “synchronous ACI”. This is a special variant of the out-of-process ACI that offers excellent performance but imposes certain restrictions on the client and is intended for advanced users.

Conclusion

This document provides an overview of ACI’s power and easy usage. The next step is to go through demos bundled with the ACI installation and consult ACI reference (accessible through Start Menu).