

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

ETRAGE Automation COM Interface (ACI) for Pro/INTRALINK Tutorial	1
Introduction	1
Out-of-Process ACI	2
ACI'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
Connecting to Commonsplace – ICommonspace.....	4
Finding the Object in Commonsplace – ICommonspaceProductItem.....	4
Checking-out the Part – IWorkspace and IWorkspaceProductItem	4
Checking-in the Part	5
Disconnecting from ACI	5
Summary	6
Conclusion	6

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/INTRALINK, as its name suggests, is the **COM server that encapsulates PTC's Toolkit API's for Pro/INTRALINK**. It implements a set of COM objects that encapsulate internal Pro/INTRALINK objects (such as Commonsplace, Product Item or Workspace), so operations that client² invokes while working with these COM objects get translated to native Pro/INTRALINK APIs (Pro/TOOLKIT).

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

ACI takes care of many small technical quirks in Pro/INTRALINK Toolkit and exposes very user-friendly API (compared to Pro/INTRALINK Toolkit), so

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

² That is, client program.

application developer can get right down to business very quickly, with smooth learning curve.

Out-of-Process ACI

ACI is implemented as “out-of-process” COM server, which means it executes as a separate process and is implemented as EXE (unlike in-process COM server that is executed within address space of client process and is physically implemented as DLL).

This makes it well suitable for server-side programming, such as ASP.NET applications.

ACI’s Object Model

On Interfaces and Classes

ACI for Pro/INTRALINK 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.

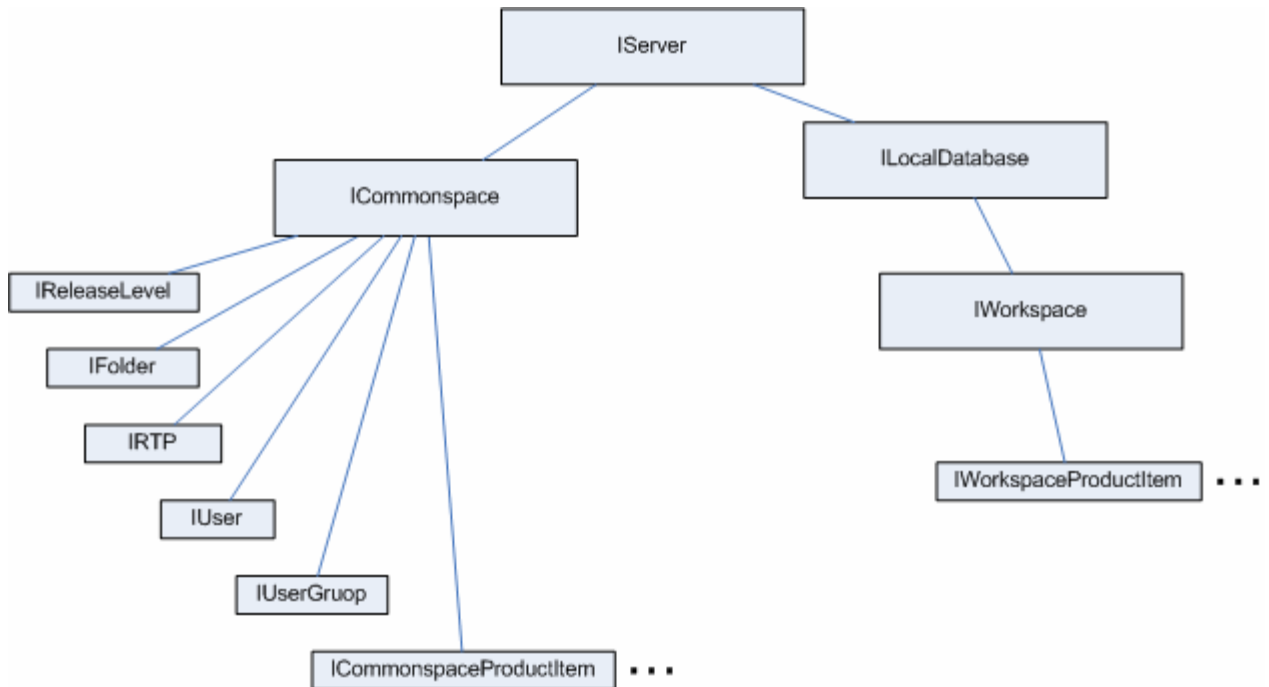
COM supports multiple interfaces per class, but ACI deliberately implements exactly one (client visible) interface per class, to avoid confusion that might arise otherwise. If class has name “CSomeName”, its interface’s name will be always be in form “ISomeName”.

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 are connected to Pro/INTRALINK Commonsense and Workspace, run-time ACI objects might form aggregation hierarchy similar to this:

³ The only situation in which the client would access a class directly is initial instantiation of CServer 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.



This tutorial will illustrate how to programmatically access (part of) this hierarchy, and should give you a decent idea of the basic “philosophy” of how ACI is used.

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 `IntralinkServer` 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 client can do anything with ACI, it must instantiate `CServer` 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 IntralinkServer.CServer
```

This line of code will effectively start the EXE that implements ACI. You can use Windows Task Manager to see that `et_ils.exe` is started and in most configurations, `BAT`⁴ file console window will also become visible.

`IServer` represents the COM server itself not Pro/INTRALINK server. It contains information, such as location of debug log, which is not directly related to Pro/INTRALINK. Its main purpose is to connect to Pro/INTRALINK Commonsense and Workspace and support objects that represent them (`ICommonspace` and `IWorkspace`).

Connecting to Commonsense – ICommonspace

Once you got hold of `IServer`, you can connect to Pro/INTRALINK server itself through `IServer`'s sub-object:

```
If Not server.Commonspace.Connect("username", "password") Then
    MsgBox("Could not connect to Commonsense.")
    Return
End If
```

This will physically connect to Pro/INTRALINK server. If for any reason (such as invalid user credentials) connection cannot be established, `Connect()` method will return `False`.

Exact details of connection (such as Pro/INTRALINK server host) are configured during the ACI installation.

Finding the Object in Commonsense – ICommonspaceProductItem

Parts, assemblies, drawings and other objects stored in Pro/INTRALINK are collectively known as “product items”. Following code finds the part of given name that is stored inside Pro/INTRALINK Commonsense:

```
Dim pi As IntralinkServer.ICommonspaceProductItem = _
    server.Commonspace.ProductItems.CollectOneByName("my_part.prt")
If pi Is Nothing Then
    MsgBox("Product item not found.")
    Return
End If
```

Note that `CollectOneByName()` method will return `Nothing` as a result if given name does not identify existing product item within the Commonsense.

Checking-out the Part – IWorkspace and IWorkspaceProductItem

Now that we hold reference to product item, we can check it out by:

⁴ Technically, ACI EXE is not started directly, but through `BAT` file that sets-up environment necessary to find all required DLLs and other resources. This `BAT` file is created during the ACI installation and “points” to existing Pro/INTRALINK Toolkit Environment that is configured for particular Pro/INTRALINK server.

```

Dim workspace As IntralinkServer.IWorkspace = _
    server.LocalDatabases.Default.Workspaces.Default
Dim wspi As IntralinkServer.IWorkspaceProductItem = _
    pi.CheckOut(workspace)
MsgBox("Product item checked out to: " + wspi.PathForRead)

```

First line simply gets the default Workspace of default Local Database. We could have chosen to create our own Workspace/Database or to connect to the existing one, but for the purpose of this tutorial we will let ACI create one for us.

Second line effectively checks-out the product item into Workspace we obtained in the previous line.

Third line shows the path of created local file. You can now load the part from that path into Pro/ENGINEER and modify it.

Checking-in the Part

Let's say you modified the part at `wspi.PathForRead` and you want to check-in the changes back to the Pro/INTRALINK Commonsense. All you need to do is:

```
workspace.CheckIn()
```

ACI will automatically detect which files in Workspace are modified and will perform check-in on them.

Disconnecting from ACI

ACI will automatically shut-down when your VB.NET applications 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 Commonsense or Workspace connections 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
pi = Nothing
workspace = Nothing
wspi = 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 check-out the part from Pro/INTRALINK and then check it back to Pro/INTRALINK looks like this:

```
Dim server As New IntralinkServer.CServer

If Not server.Commonspace.Connect("username", "password") Then
    MsgBox("Could not connect to Commonsplace.")
    Return
End If

Dim pi As IntralinkServer.ICommonspaceProductItem = _
    server.Commonspace.ProductItems.CollectOneByName("my_part.prt")
If pi Is Nothing Then
    MsgBox("Product item not found.")
    Return
End If

Dim workspace As IntralinkServer.IWorkspace = _
    server.LocalDatabases.Default.Workspaces.Default
Dim wspi As IntralinkServer.IWorkspaceProductItem = _
    pi.CheckOut(workspace)
MsgBox("Product item checked out to: " + wspi.PathForRead)

' (Do something useful with 'wspi'.)

workspace.CheckIn()
```

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).