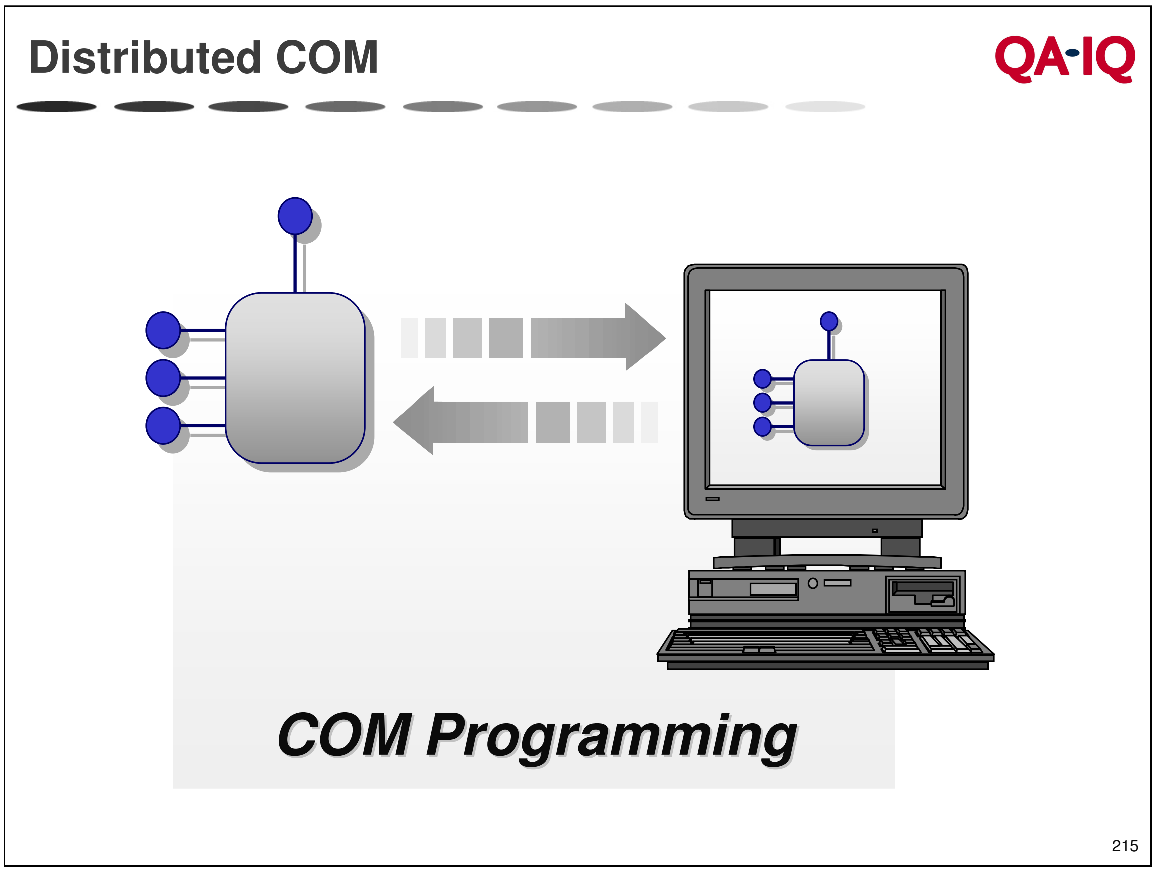
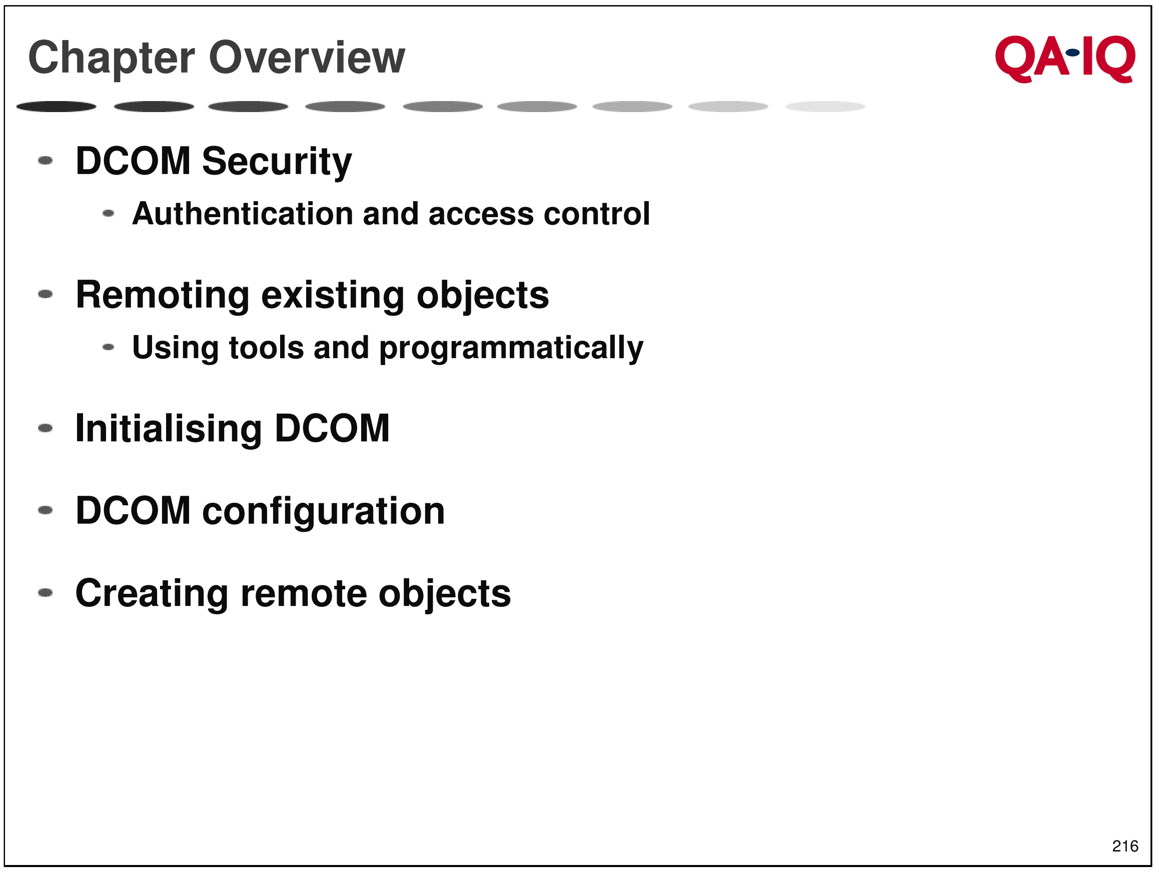


Distributed COM

COM Programming

215



Chapter Overview

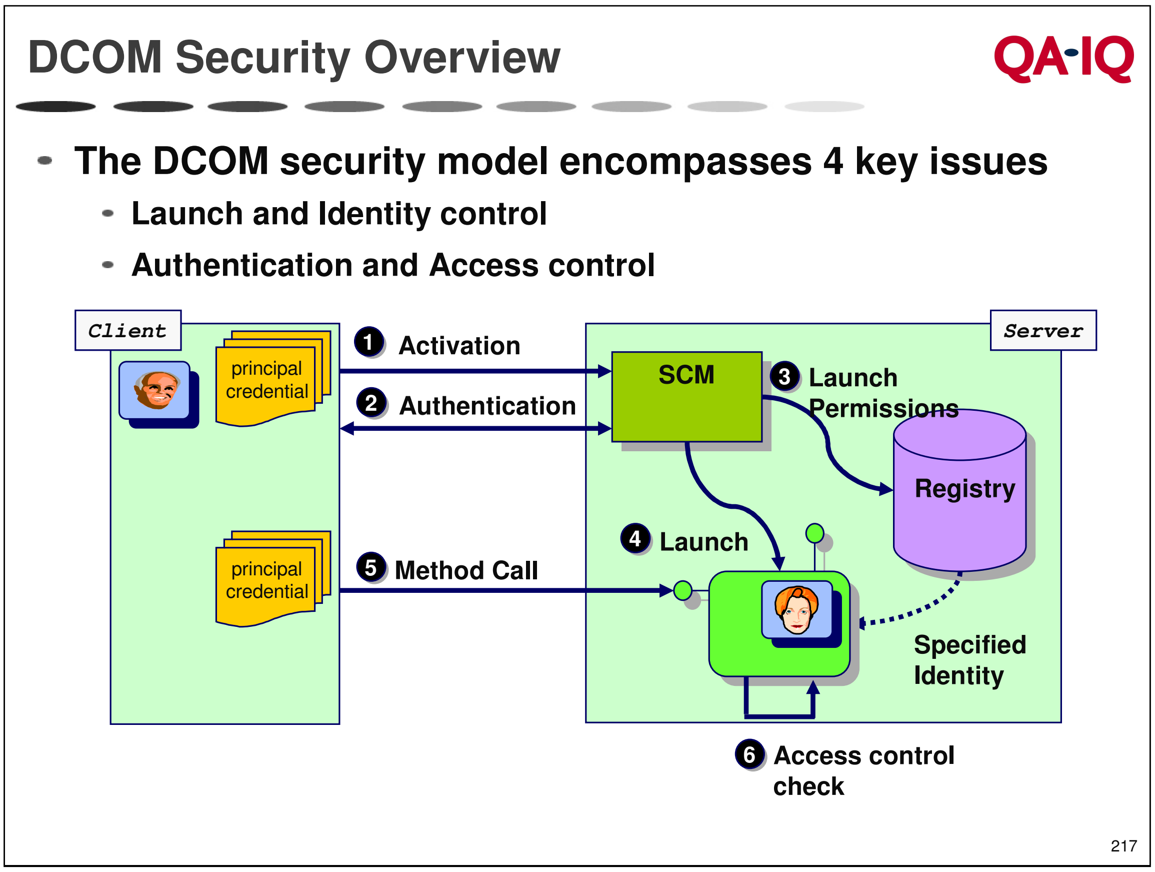
QA-IQ

- DCOM Security
  - Authentication and access control
- Remoting existing objects
  - Using tools and programmatically
- Initialising DCOM
- DCOM configuration
- Creating remote objects

216

This chapter looks at Distributed COM, a technology that allows COM objects to be instantiated on a separate machine from the client. Existing COM objects can be remoted in this way.

In addition, when DCOM arrived it provided us with a new threading model to allow high performance objects to be deployed in a business environment. The chapter will look at the new facilities that DCOM provides as well as some of the tools for managing DCOM objects.



DCOM Security Overview

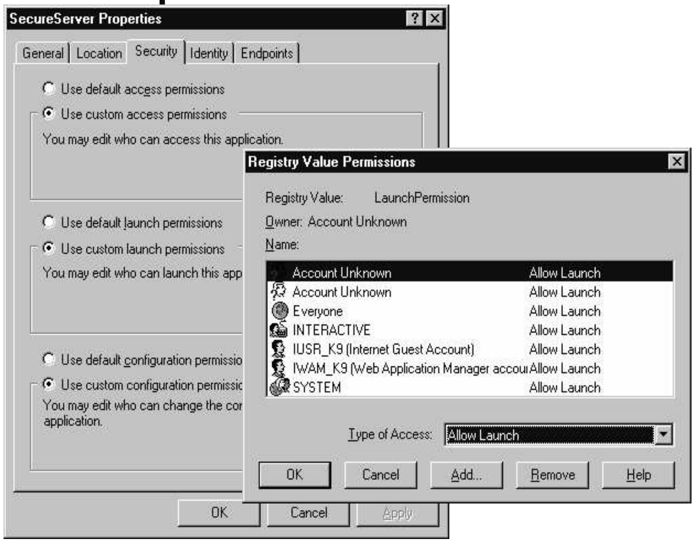
- The DCOM security model encompasses 4 key issues
  - Launch and Identity control
  - Authentication and Access control

Launch control determines who is allowed to launch DCOM servers. Assuming a particular user is allowed to launch a particular server, identity control then specifies the account under which the DCOM server should be launched. This provides the server with a set of security credentials. Authentication control is used firstly to identify a particular caller and then to ensure that the network transmission is authentic (from the appropriate caller), confidential (preventing anyone else from tapping into the conversation) and finally to validate the message’s integrity - to ensure that message has not been tampered with en-route. The particular level of authentication can be specified per machine, per server or programmatically per interface. In the above sequence:

(1) A client process makes a COM activation request. The client process will be running with a specific set of security credentials. These may or may not be used for the activation request. As we shall see, it is possible to specify an alternate principal to make the activation request, via parameters supplied to CoCreateInstanceEx().
(2) An authentication process aims to identify the caller.
(3) The SCM on the server machine accepts the inbound activation request and consults the registry for the relevant launch permissions.
(4) Assuming the caller has suitable launch permissions for the server in question, the SCM will launch the server using the identity specified in the registry. This may be the interactive user, the launching user or preferably a nominated user account set-up specifically for this purpose.
(5) The client now makes a method call.
(6) The DCOM server either accepts or rejects the call based on the enforced security policy established programmatically via CoInitializeSecurity(), or which has been adjusted for a specific proxy.

Launch Control

- **Launch permissions are used to determine which users can start server processes**



SecureServer Properties

General | Location | Security | Identity | Endpoints

( ) Use default access permissions
(•) Use custom access permissions
You may edit who can access this application.

( ) Use default launch permissions
(•) Use custom launch permissions
You may edit who can launch this app

( ) Use default configuration permissio
(•) Use custom configuration permissic
You may edit who can change the cor application.

OK | Cancel | Apply

Registry Value Permissions

Registry Value: LaunchPermission
Owner: Account Unknown
Name:

Account Unknown	Allow Launch
Account Unknown	Allow Launch
Everyone	Allow Launch
INTERACTIVE	Allow Launch
IUSR_K9 (Internet Guest Account)	Allow Launch
IWAM_K9 (Web Application Manager accou	Allow Launch
SYSTEM	Allow Launch

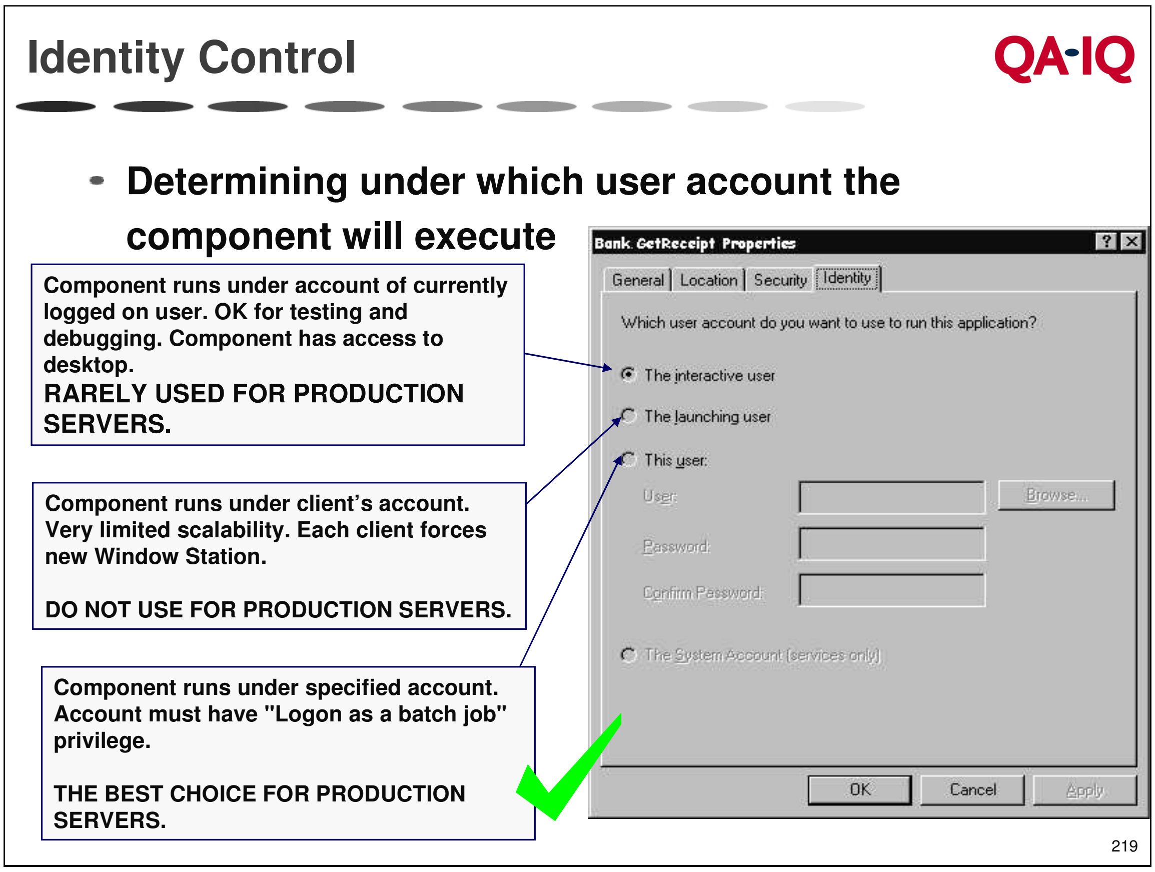
Type of Access: Allow Launch

OK | Cancel | Add... | Remove | Help

218

Using DCOMCNFG you can specify exactly which users are allowed to launch DCOM servers. You can establish a set of per-machine defaults that will be used in the absence of component specific settings, or you can tailor the settings for a particular component.

Under the covers, DCOMCNFG actually creates a Win32 `SECURITY_DESCRIPTOR` structure based on the information you supply via the graphical interface. A serialised (flattened using offsets instead of pointers) version of this structure is then written to the registry.



Identity Control

QA-IQ

- **Determining under which user account the component will execute**

Component runs under account of currently logged on user. OK for testing and debugging. Component has access to desktop.
RARELY USED FOR PRODUCTION SERVERS.

Component runs under client's account. Very limited scalability. Each client forces new Window Station.

DO NOT USE FOR PRODUCTION SERVERS.

Component runs under specified account. Account must have "Logon as a batch job" privilege.

THE BEST CHOICE FOR PRODUCTION SERVERS.

Bank.GetReceipt Properties

General | Location | Security | Identity

Which user account do you want to use to run this application?

[✓ The interactive user]
The launching user
This user:
User:
Password:
Confirm Password:
The System Account (services only)

219

Specifying that your COM server should run as "*The interactive user*", should only be considered for testing and debugging purposes. Beyond that, there are a number of potential pitfalls with this option. Firstly, this relies on there being an interactive user logged on in the first place - not the usual state of affairs for a server. Secondly, you can not predict ahead of time what access rights your server will be granted. This of course depends on which particular user is currently logged on. Thirdly, if the interactive user logs off while your server process is running, it will be terminated rather prematurely.

"*The launching user*" (which is the default option), presents further problems. This approach causes your server to run with the client's security token. For this option to work, your server must be able to obtain a meaningful token for the client and so you must be able to authenticate the client. Each client will force a new instance of your COM server to be launched. The reason for this is that each server must run under the client's access token, so User A and User B can not possibly share the same instance of a single server. This approach results in NT creating a new **invisible** Window Station (a security environment) for each server instance. Unfortunately Window stations are a limited resource under NT. By default, you are limited to around 16 or so Window stations per machine, although this number can be increased with various registry tweaks.

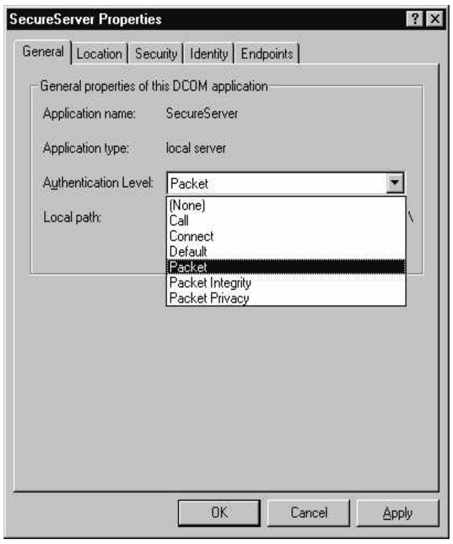
The recommended approach is to inform the SCM up front about a distinguished principal whose token should be used to launch your server, no matter who the activator is. Unauthenticated clients can call the server, and the server can access both local and remote resources. Specifying this setting in `DCOMCNFG` results in a `RunAs` named value being created under the servers `AppID` key. The associated password is stored in a secure area of the registry known as the Local Security Authority (LSA). The `DCOMCNFG` tool uses the LSA API to set the `RunAs` secret. This is one of the reasons why `DCOMCNFG` requires Administrator privileges.

Be aware that the account specified as the `RunAs` user requires the "`logon as a batch job`" privilege. This must be granted on the machine where the server runs.

Authentication

QA-IQ

- **Determining the caller's identity**


SecureServer Properties

General properties of this DCOM application

Application name: SecureServer

Application type: local server

Authentication Level: Packet

(None)
Call
Connect
Default
Packet
Packet Integrity
Packet Privacy

Local path:

220

Authentication refers to the process of establishing a caller's identity. This of course is essential in order to be able to implement any form of access control policy. There are a number of ways to specify the required level of authentication. `DCOMCNFG` allows machine wide and component specific authentication levels to be established. Furthermore, a process can set an application wide authentication level using the `CoInitializeSecurity()` API. Finally, it is possible for a client to affect the authentication level used on an interface by interface basis, using the `IClientSecurity` interface.

While packet privacy provides the most secure mode of authentication, with each packet being authenticated, tamper checked and encrypted, this does not come without a significant performance penalty. For this reason it is common for an application to set a lower default authentication level (typically connect), with a specific interface having its authentication level increased for secure transmissions.

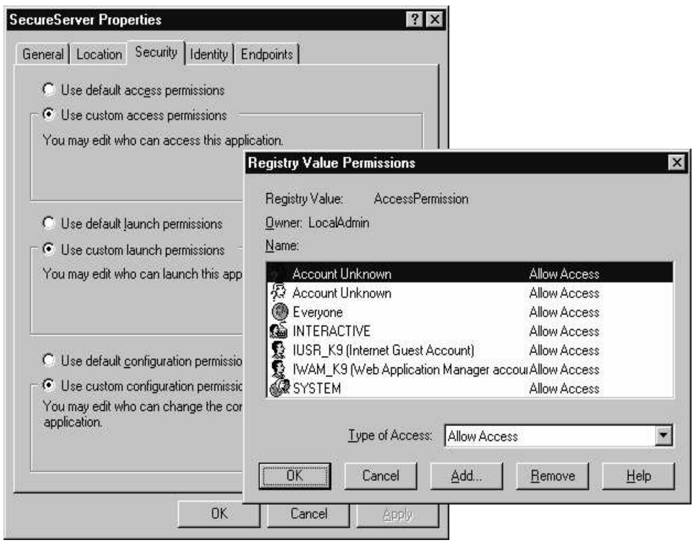
If you don't establish component wide settings explicitly (via `CoInitialiseSecurity()`), COM will use established defaults. Under NT4 SP4 `DCOMCNFG` allows you specify component specific defaults, established via the `AuthenticationLevel` named value beneath the `[HKCR/AppID]` key. SP3 and earlier only allow you to establish a single machine wide default applicable to all components.

You should also be aware that requested levels can be silently promoted. For example on a UDP connection, there is no concept of a connection, and so connect level will be promoted to call. However, as Microsoft RPC does not currently implement call-level authentication, this is promoted to packet.

Access Control

QA-IQ

- **Determining the if caller is allowed access**



Use default access permissions [✓ Use custom access permissions]

You may edit who can access this application.

Use default launch permissions [✓ Use custom launch permissions]

You may edit who can launch this app

Use default configuration permissio [✓ Use custom configuration permissic]

You may edit who can change the cor application.

Registry Value: AccessPermission

Owner: LocalAdmin

Name:

Account Unknown	Allow Access
Account Unknown	Allow Access
Everyone	Allow Access
INTERACTIVE	Allow Access
IUSR_K9 (Internet Guest Account)	Allow Access
IWAM_K9 (Web Application Manager accou	Allow Access
SYSTEM	Allow Access

Type of Access: Allow Access

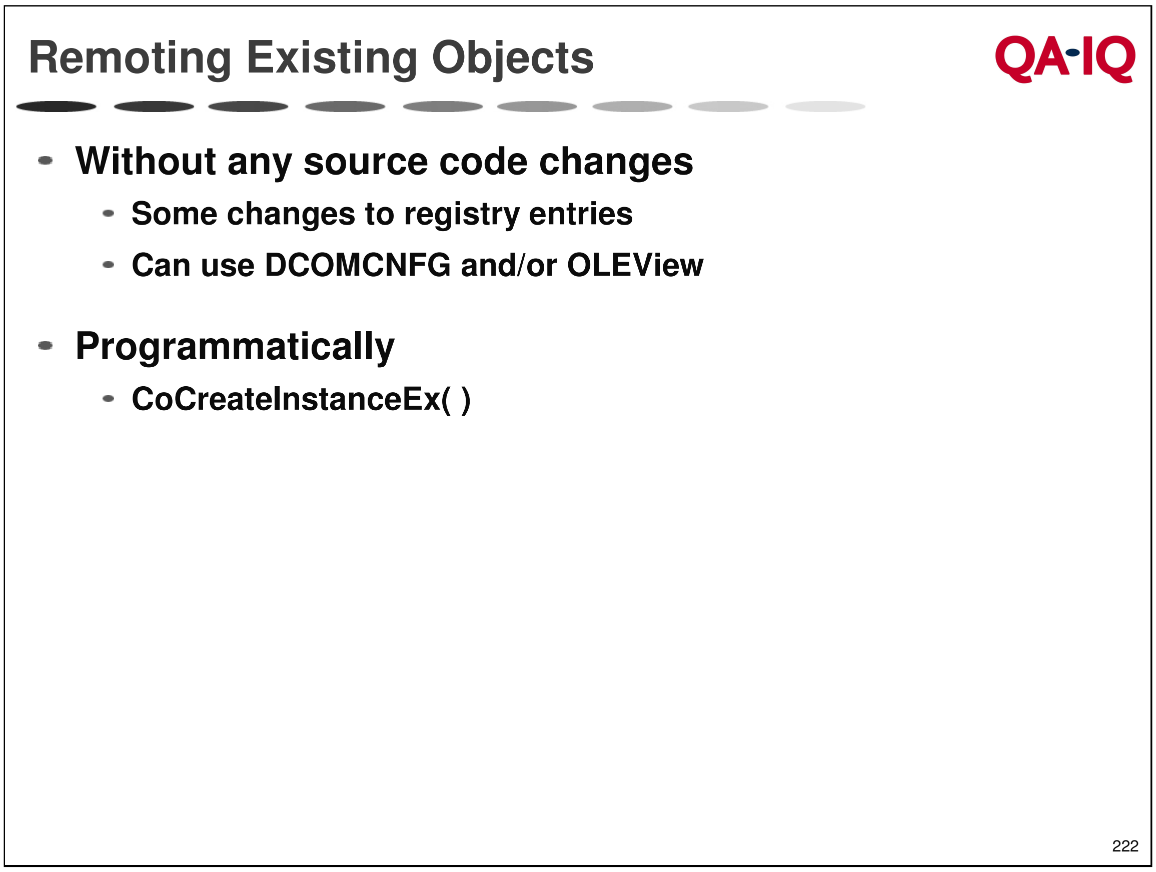
221

Using DCOMCNFG you can specify exactly which users are allowed to access DCOM servers. You can establish a set of per-machine defaults that will be used in the absence of component specific settings, or you can tailor the settings for a particular component. DCOMCNFG under the covers actually creates a Win32 `SECURITY_DESCRIPTOR` structure based on the information you supply via the graphical interface. A serialised (flattened using offsets instead of pointers) version of this structure is then written to the registry.

Using programmatic security it is possible to override the declarative defaults at run time.

NB: One important point to be aware of. You will notice when you first begin to use DCOMCNFG, that the built-in SYSTEM account is always granted access permissions to all components. Do not be tempted to remove this entry, as the COM SCM (implemented within the rpcss.exe process) runs under the SYSTEM account, and as a result SYSTEM **must** have access permissions. Failure to grant the SYSTEM account access permissions results in the rather meaningless (and plain wrong!) return code of `E_OUTOFMEMORY`. You have been warned!

Contrary to certain documentation, COM does not require the SYSTEM account to be granted launch permissions to the component.



Remoting Existing Objects

QA-IQ

- Without any source code changes
  - Some changes to registry entries
  - Can use DCOMCNFG and/or OLEView
- Programmatically
  - CoCreateInstanceEx( )

222

COM allows you to remote an existing local server onto a remote machine either by modifying the registry (declarative approach) or under programmatic control. Programmatic settings always override registry settings.

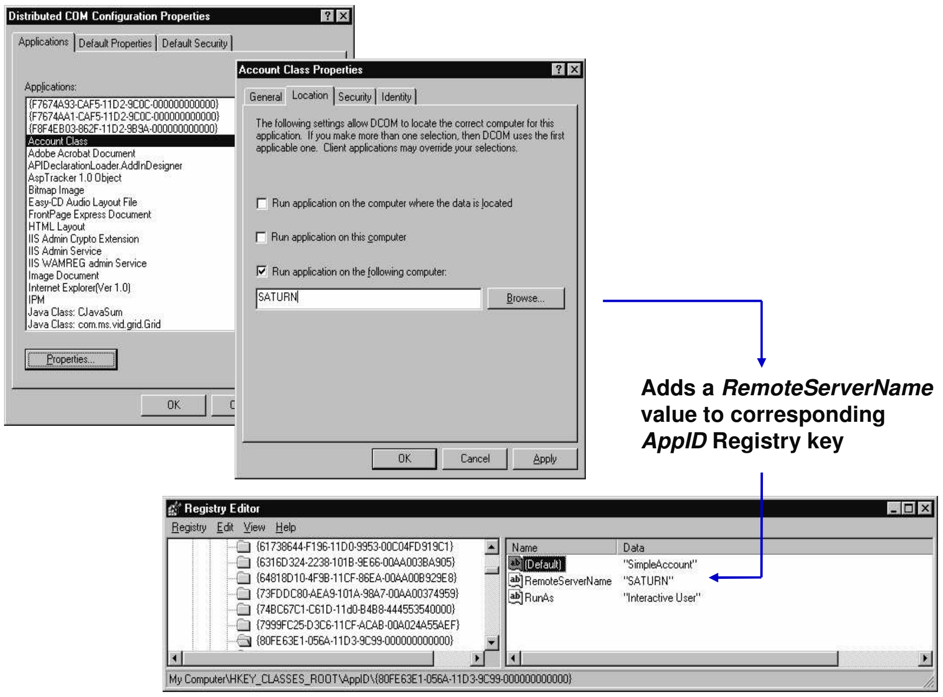
Microsoft recommend that developers should use the DCOMCNFG utility to modify registry settings rather than write explicit code. When remoting a COM object it can be quite tricky to get the security settings correct and it is much easier if you avoid coding this information.

The declarative approach is being strongly encouraged by Microsoft with MTS and COM+.

One advantage of the programmatic approach is that once you've got it right, an administrator cannot change settings for your COM object through the registry - leaving you in control.

Using DCOMCNFG on Local Machine

QA-IQ


Distributed COM Configuration Properties

Applications | Default Properties | Default Security

Applications:

{F7674A93-CAF5-11D2-9C0C-000000000000}
{F7674AA1-CAF5-11D2-9C0C-000000000000}
{F8F4EB03-862F-11D2-9B9A-000000000000}
Account Class
Adobe Acrobat Document
APIDeclarationLoader.AddInDesigner
AspTracker 1.0 Object
Bitmap Image
Easy-CD Audio Layout File
FrontPage Express Document
HTML Layout
IIS Admin Crypto Extension
IIS Admin Service
IIS WAMREG admin Service
Image Document
Internet Explorer(Ver 1.0)
IPM
Java Class: CJavaSum
Java Class: com.ms.vid.grid.Grid

Account Class Properties

General | Location | Security | Identity

The following settings allow DCOM to locate the correct computer for this application. If you make more than one selection, then DCOM uses the first applicable one. Client applications may override your selections.

☐ Run application on the computer where the data is located

☐ Run application on this computer

☑ Run application on the following computer:

SATURN

Registry Editor

{61738644-F196-11D0-9953-00C04FD919C1}
{6316D324-2238-101B-9E66-00AA003BA905}
{64818D10-4F9B-11CF-86EA-00AA00B929E8}
{73FDDC80-AEA9-101A-98A7-00AA00374959}
{74BC67C1-C61D-11d0-B4B8-444553540000}
{7999FC25-D3C6-11CF-ACAB-00A024A55AEF}
{80FE63E1-056A-11D3-9C99-000000000000}

Name	Data
(Default)	"SimpleAccount"
RemoteServerName	"SATURN"
RunAs	"Interactive User"

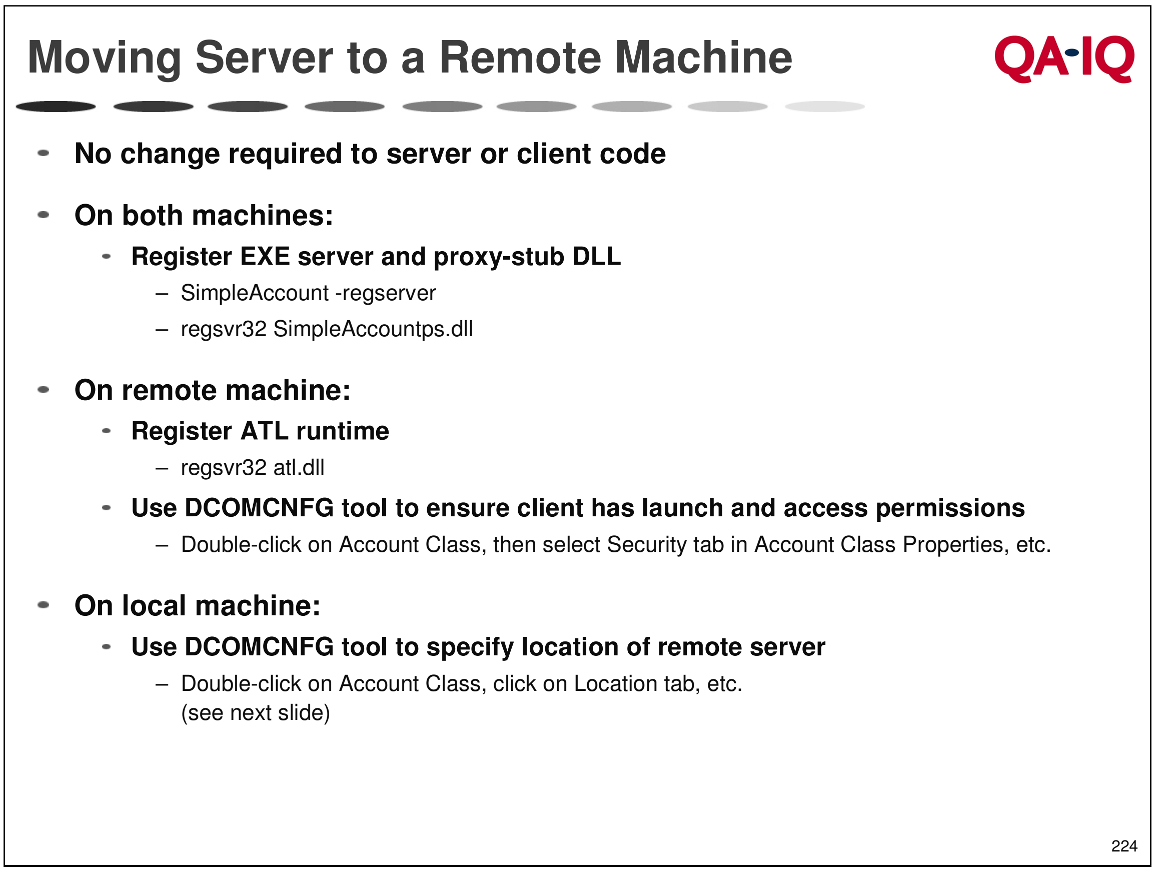
My Computer\HKEY_CLASSES_ROOT\AppID\{80FE63E1-056A-11D3-9C99-000000000000}

Adds a *RemoteServerName* value to corresponding *AppID* Registry key

223

On the local (client) machine, we use *DCOMCNFG* to specify the location of the remote server. Because the EXE server has been registered on this machine, we should find an entry for our component (e.g. *Account Class*) under the list of applications. Double-clicking on this item brings up a *Properties* dialog with four tabs. To specify the location(s) of the server (application), we click on the *Location* tab. We want the server to run *only* on a specific machine, so we deselect *Run application on this computer*, select *Run application on the following computer*, and enter the name of the remote computer. After we've clicked the OK button, if we look in the Registry, we should find an appropriate `RemoteServerName` entry under the server's *AppID* key.

Now we're ready to test. If the server is installed on a Windows NT machine, all we need to do is to run the client application as if we're using a local server. What happens is that when the client calls `CoGetClassObject()` (either directly or via `CoCreateInstance()`), if the specified context is `CLSCTX_LOCAL_SERVER`, the CLSID entry in the Registry is checked to see if it has an *AppID* entry. If so, the *AppID* key is checked to see whether it has a `RemoteServerName` entry, and if it does, an attempt is made to launch the server on the specified remote machine.



Moving Server to a Remote Machine

QA-IQ

- No change required to server or client code
- On both machines:
  - Register EXE server and proxy-stub DLL
    - SimpleAccount -regserver
    - regsvr32 SimpleAccountps.dll
- On remote machine:
  - Register ATL runtime
    - regsvr32 atl.dll
  - Use DCOMCNFG tool to ensure client has launch and access permissions
    - Double-click on Account Class, then select Security tab in Account Class Properties, etc.
- On local machine:
  - Use DCOMCNFG tool to specify location of remote server
    - Double-click on Account Class, click on Location tab, etc. (see next slide)

224

Without making any change to the server or client code, you can move the EXE server to another machine. This is possible though the magic of DCOM (Distributed COM), which is supported by Windows NT 4.0 and later, Windows ME, Windows 98 or Windows 95 with the DCOM add-in.

Note that you must register the EXE server on both the client and the server machines. Obviously this is necessary on the server machine, but why do we have to do it on the client machine? The answer is that the client machine needs the *CLSID* and *AppID* information. In fact, after performing this registration, you can remove the EXE server code from the client machine. If you want to use standard marshaling, you must also register the MIDL-generated proxy/stub DLL on both machines.

You will also need to ensure that the ATL runtime (`atl.dll`) is installed in the `System32` (Windows NT), or `System` (Windows 95), directory on both machines. You can find this DLL on the Visual Studio or Visual C++ installation CD. (Note that with version 5.0, two versions of `atl.dll` were provided: one for Windows NT and the other for Windows 95.) You'll also need to make sure that `atl.dll` is registered on both machines.

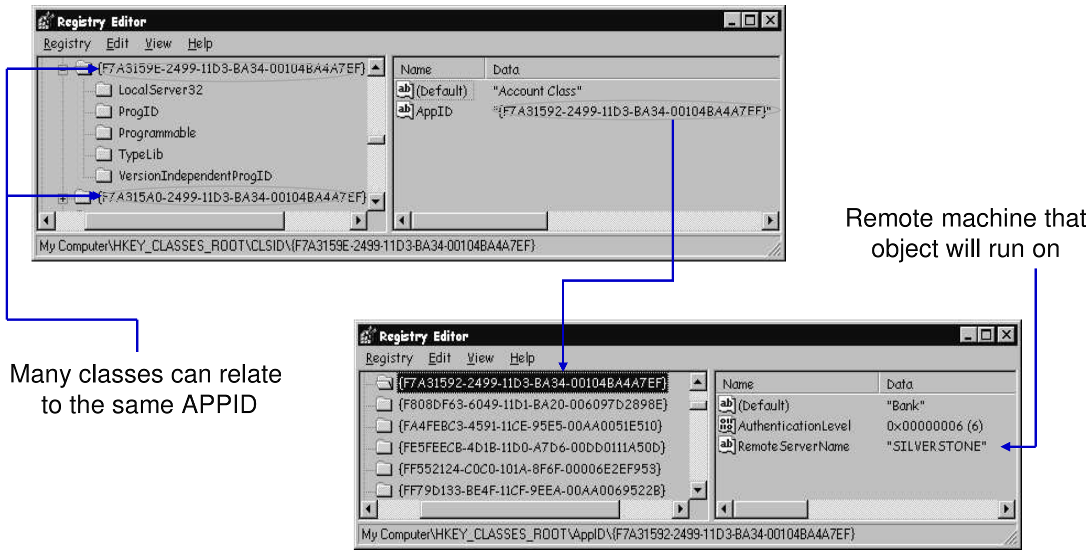
Next, on the remote (server) machine, use the *DCOMCNFG* configuration tool to ensure that the client has permission to launch (only possible with Windows NT) and access the remote object.

Then, on the local (client) machine, use *DCOMCNFG* to specify the location of the remote server. This information is stored under the *AppID* key in the Registry. We'll look at this in a little more detail on the next slide.

The APPID Registry Key

QA-IQ

- **DCOM introduces the APPID key**
  - **To maintain per-server settings**


Registry Editor
Registry Edit View Help
Many classes can relate to the same APPID
Remote machine that object will run on
Registry Editor
Registry Edit View Help

```
{F7A3159E-2499-11D3-BA34-00104BA4A7EF}
    LocalServer32
    ProgID
    Programmable
    TypeLib
    VersionIndependentProgID
{F7A315A0-2499-11D3-BA34-00104BA4A7EF}
```

Name	Data
(Default)	"Account Class"
AppID	"{F7A31592-2499-11D3-BA34-00104BA4A7EF}"

My Computer\HKEY_CLASSES_ROOT\CLSID\{F7A3159E-2499-11D3-BA34-00104BA4A7EF}

```
{F7A31592-2499-11D3-BA34-00104BA4A7EF}
{F808DF63-6049-11D1-BA20-006097D2898E}
{FA4FEBC3-4591-11CE-95E5-00AA0051E510}
{FE5FEECB-4D1B-11D0-A7D6-00DD0111A50D}
{FF552124-C0C0-101A-8F6F-00006E2EF953}
{FF79D133-BE4F-11CF-9EEA-00AA0069522B}
```

Name	Data
(Default)	"Bank"
AuthenticationLevel	0x00000006 (6)
RemoteServerName	"SILVERSTONE"

My Computer\HKEY_CLASSES_ROOT\AppID\{F7A31592-2499-11D3-BA34-00104BA4A7EF}

225

The APPID key groups together per-server (executable) settings. Each COM class supported by a particular server references its associated server settings via the APPID named value beneath the [HKCR\CLSID\<CLSID>] key. This contains a GUID which points to the [HKCR\APPID\<APPID_GUID>] key. The various named values that are supported beneath this APPID key are discussed on a subsequent slide.

It is important to realise that some settings apply only to client machines, and some only to server machines. One of the most fundamental entries on the client side is RemoteServerName, which is used to name the remote machine upon which the server should be instantiated. (This assumes the server and proxy-stub DLLs have been correctly registered on the nominated server).

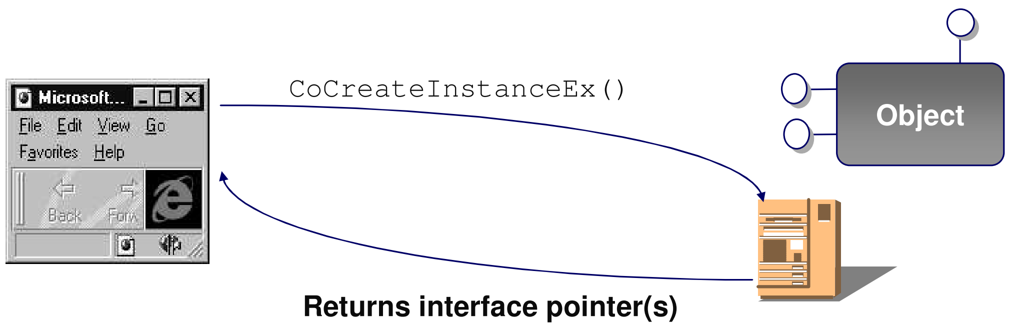
If the client activation request explicitly specifies CLSCTX_REMOTE_SERVER, the activation request will be forwarded directly to the nominated server. For legacy servers it is more likely that the CLSCTX_LOCAL_SERVER class context be specified. In this event, *providing there is no local server process registered*, the activation request will still be forwarded to the remote machine.

An executable file name (for example "BankSvr.EXE" may also be registered under the [HKCR\APPID] key. These entries are used to specify default access permissions for the server. (For example none or only some of the server's classes may have registered an APPID GUID as a named value under [HCKR\CLSID\<CLSID>].

Remoting Programmatically

QA-IQ

- CoCreateInstanceEx()
  - Similar to CoCreateInstance()
- Takes a COSERVERINFO structure
  - Specifies name of machine on which to instantiate the object
  - CoGetClassObject() takes a COSERVERINFO structure
- Also takes an array of MULTI_QI structures

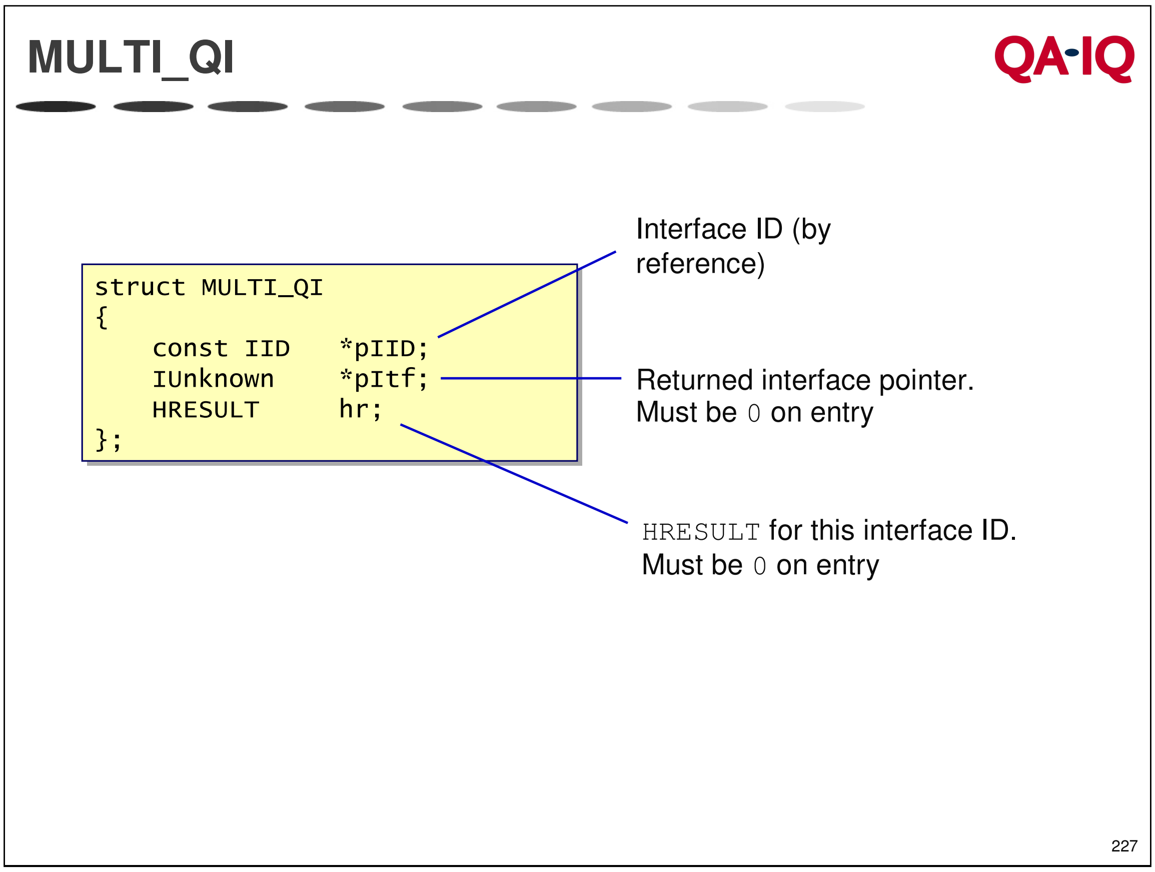


226

The first new API is `CoCreateInstanceEx()`. This API is similar to `CoCreateInstance()`. However it takes a `COSERVERINFO*` as one of its parameters. One field of this structure that is used is `pwszName`, which points to a UNICODE string containing the name or IP address of the machine on which the object will be instantiated.

`CoGetClassObject()` also takes a `COSERVERINFO*` as its third parameter.

`CoCreateInstanceEx()` also takes an array of `MULTI_QI` structures allowing clients to request multiple interface pointers at object creation. This reduces network bandwidth requirements as it results in fewer remote calls.



MULTI_QI

```
struct MULTI_QI
{
    const IID    *pIID;
    IUnknown     *pItf;
    HRESULT      hr;
};
```

Interface ID (by reference)

Returned interface pointer. Must be `0` on entry

`HRESULT` for this interface ID. Must be `0` on entry

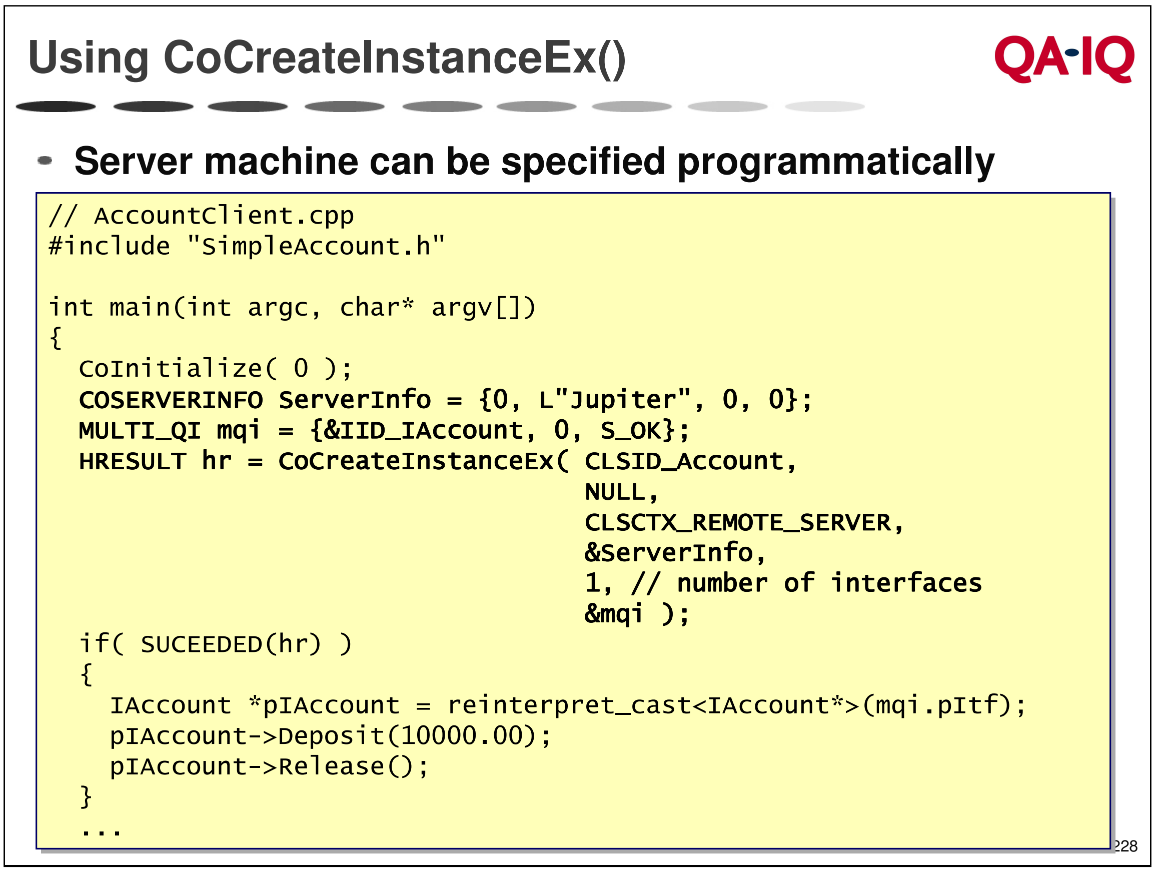
227

The `MULTI_QI` structure contains the interface ID of the interface that is being requested, a placeholder for storing the returned interface pointer and a placeholder for an `HRESULT`. The interface pointer must be `NULL` and the `HRESULT` set to `0` on entry.

An array of these structures can be passed to the new `CoCreateInstanceEx()` call to enable a single call to fetch multiple references to the object with a single network call. This is much more efficient than making multiple round trips, which would be required with `CoCreateInstance()`. Consequently, each `MULTI_QI` structure must have its own `HRESULT`, as the requested interface might not be supported on the object.

`CoCreateInstanceEx()` will return `S_OK` if all requested interfaces are successfully returned, and `CO_S_NOTALLINTERFACES` if only selected interfaces are available. In this latter case, you will need to check each of the `MULTI_QI` `hr` values in turn. `CoCreateInstanceEx()` returns `E_NOINTERFACE` if none of the interfaces are supported.

`MULTI_QI` structures can also be used with the `IMultiQI` interface, that is provided by COM on proxies.



Using CoCreateInstanceEx()

QA-IQ

- **Server machine can be specified programmatically**

```
// AccountClient.cpp
#include "SimpleAccount.h"

int main(int argc, char* argv[])
{
  CoInitialize( 0 );
  COSERVERINFO ServerInfo = {0, L"Jupiter", 0, 0};
  MULTI_QI mqi = {&IID_IAccount, 0, S_OK};
  HRESULT hr = CoCreateInstanceEx( CLSID_Account,
                                   NULL,
                                   CLSCTX_REMOTE_SERVER,
                                   &ServerInfo,
                                   1, // number of interfaces
                                   &mqi );
  if( SUCEEDED(hr) )
  {
    IAccount *pIAccount = reinterpret_cast<IAccount*>(mqi.pItf);
    pIAccount->Deposit(10000.00);
    pIAccount->Release();
  }
  ...
```

228

There is another way to run the server on another machine. Rather than using *DCOMCNFG* to change the registry settings on the client machine, we can specify the server machine programmatically by modifying the client code.

As shown on the slide, we need to call `CoCreateInstanceEx()` instead of `CoCreateInstance()`. As discussed, this function takes a pointer to a `COSERVERINFO` structure in which we specify the name of the remote server (*Jupiter* in this example).

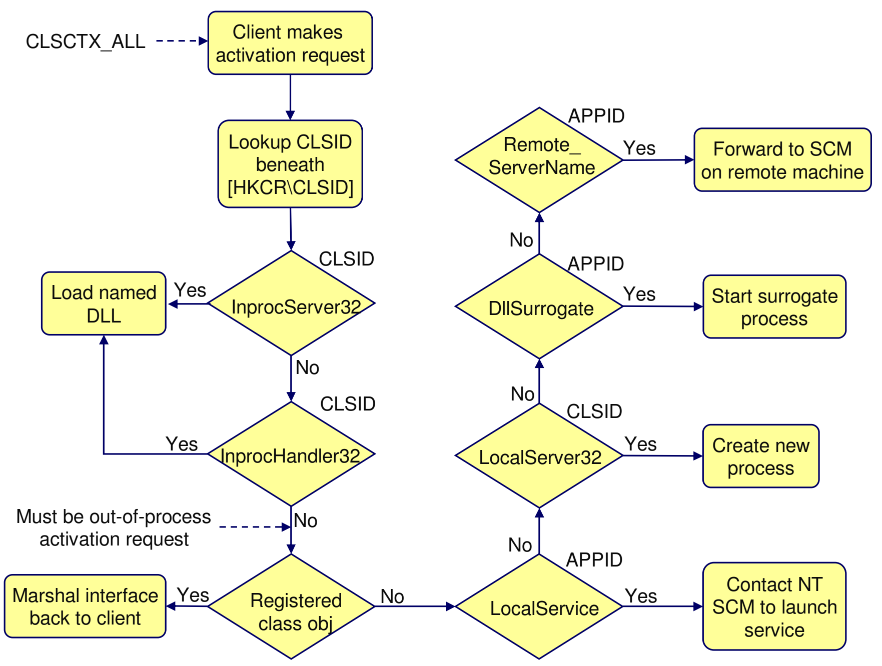
By specifying the `CLSCTX_REMOTE_SERVER` context flag, the SCM on the client machine *does not check the local registry*, but instead simply passes the creation call directly through to the server's SCM.

In the example above, we are only asking for a single interface of type IID_IAccount. We still have to pass in the requested information through an array of `MULTI_QI` structures, but in this case it is an array of length one. Given that we are only asking for a single interface we can simply check the reply of the `CoCreateInstanceEx()` call, as a successful reply code means that the interface must be present.

Finally, we need to cast the `IUnknown *` from the `pItf` member to an `IAccount *`. This seemingly dangerous downcast is acceptable because the `MULTI_QI` structure essentially makes use of the `iid_is()` attribute on the `pItf` member.

Remote Activation Steps

QA-IQ



229

This flow chart shows the sequence of steps followed by the SCM when servicing an activation request. The class context CLSCTX_ALL is assumed, which will result in the SCM looking for the most efficient way of activating the class (i.e. in-process) through to the least efficient way (remote). Other class contexts will effect the sequence of events - for example CLSCTX_REMOTE_SERVER will serve to short-circuit the sequence and cause the SCM to attempt a remote activation directly.

In response to the initial activation request, the SCM looks up the CLSID beneath [HKCR\CLSID\<CLSID>]. If this key is not located (the class has not been registered), then the HRESULT CLASS_E_CLASSNOTREGISTERED is returned.

Assuming the CLSID entry is present, the SCM looks for the InprocServer32 key, which if present names a DLL. The SCM will load the DLL. If InprocServer32 is not located, the InprocHandler32 key is looked up. If this is present it also names a DLL - this time an associated handler DLL. If this value is not present it can be assumed at this point that the activation must be out-of-process.

The SCM next consults its internal table of registered class objects looking for one with a matching CLSID.

If no registered class objects are found, the SCM will need to launch the server process. Before doing this it must determine whether the server is an NT service. It checks for the LocalService entry beneath the APPID key. If present this entry provides the name of an NT service. In this instance the COM SCM contacts the NT SCM to launch the process as a service.

If LocalService is not found, the SCM once again consults the CLSID key this time looking for the LocalServer32 sub key. If located, the SCM will start the named process using the CreateProcessAsUser() API. If this entry is not located, the SCM next looks for a DllSurrogate entry under the APPID key. If found the SCM will launch the surrogate process - either the default dllhost.exe or if one has been specified, a custom surrogate process.

Finally, if an activation has still not occurred, the SCM will look for the RemoteServerName entry under the APPID key. If present the local (client) SCM will contact the remote (server) SCM and ask it to perform the activation.

Summary

QA-IQ

- **Distributed COM allows creation of remote objects**
- **The DCOMCNFG tool allows you to configure the Registry so that you can launch and access a remote EXE server**
  - **No changes required to client or server code.**
  - **The same can be done programatically**
- **Objects can be secured**

230