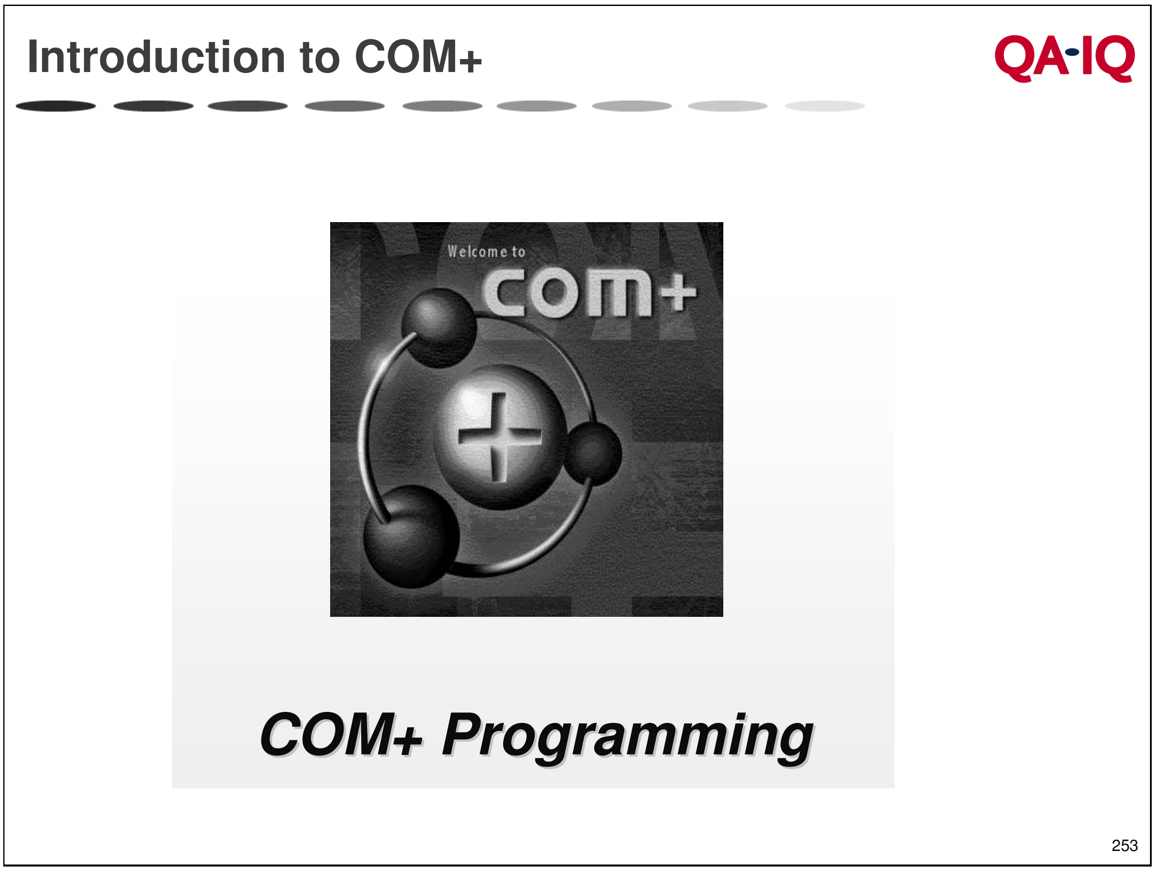
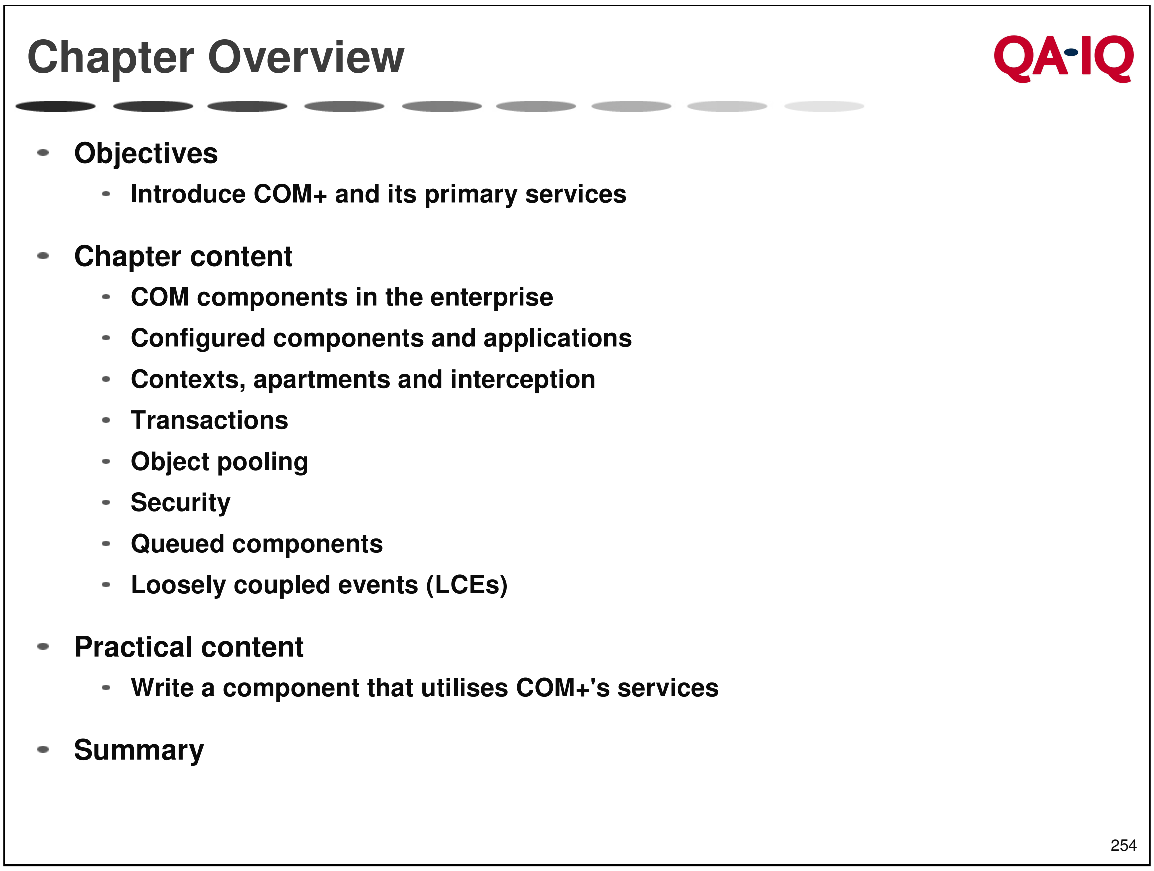


Introduction to COM+

QA-IQ

COM+ Programming



Chapter Overview

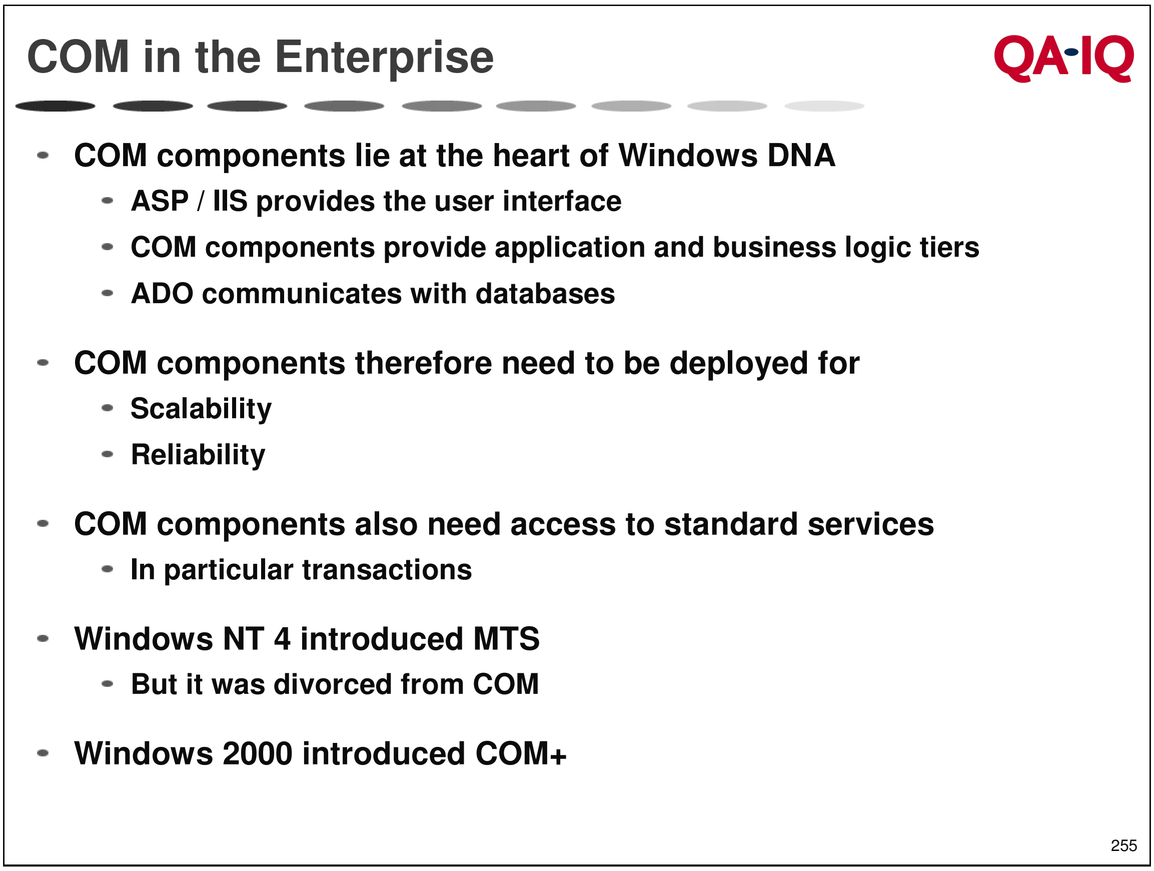
QA-IQ

- **Objectives**
  - Introduce COM+ and its primary services
- **Chapter content**
  - COM components in the enterprise
  - Configured components and applications
  - Contexts, apartments and interception
  - Transactions
  - Object pooling
  - Security
  - Queued components
  - Loosely coupled events (LCEs)
- **Practical content**
  - Write a component that utilises COM+'s services
- **Summary**

254

This chapter will provide an introduction to COM+.

Regrettably, we can only provide a brief overview of this technology. However, in this section we will identify the major services that COM+ can bring to your enterprise development projects.



COM in the Enterprise

- COM components lie at the heart of Windows DNA
  - ASP / IIS provides the user interface
  - COM components provide application and business logic tiers
  - ADO communicates with databases
- COM components therefore need to be deployed for
  - Scalability
  - Reliability
- COM components also need access to standard services
  - In particular transactions
- Windows NT 4 introduced MTS
  - But it was divorced from COM
- Windows 2000 introduced COM+

255

COM is used extensively within large scale enterprise developments.

COM components often contain the core application and business logic that is called from the top level presentation tiers (whether over Internet, Intranet or traditional client/server architectures). In turn, these COM components then utilise ADO (Active Data Objects) to communicate with stored procedures and other functionality offered by database engines such as SQL Server and Oracle Server.

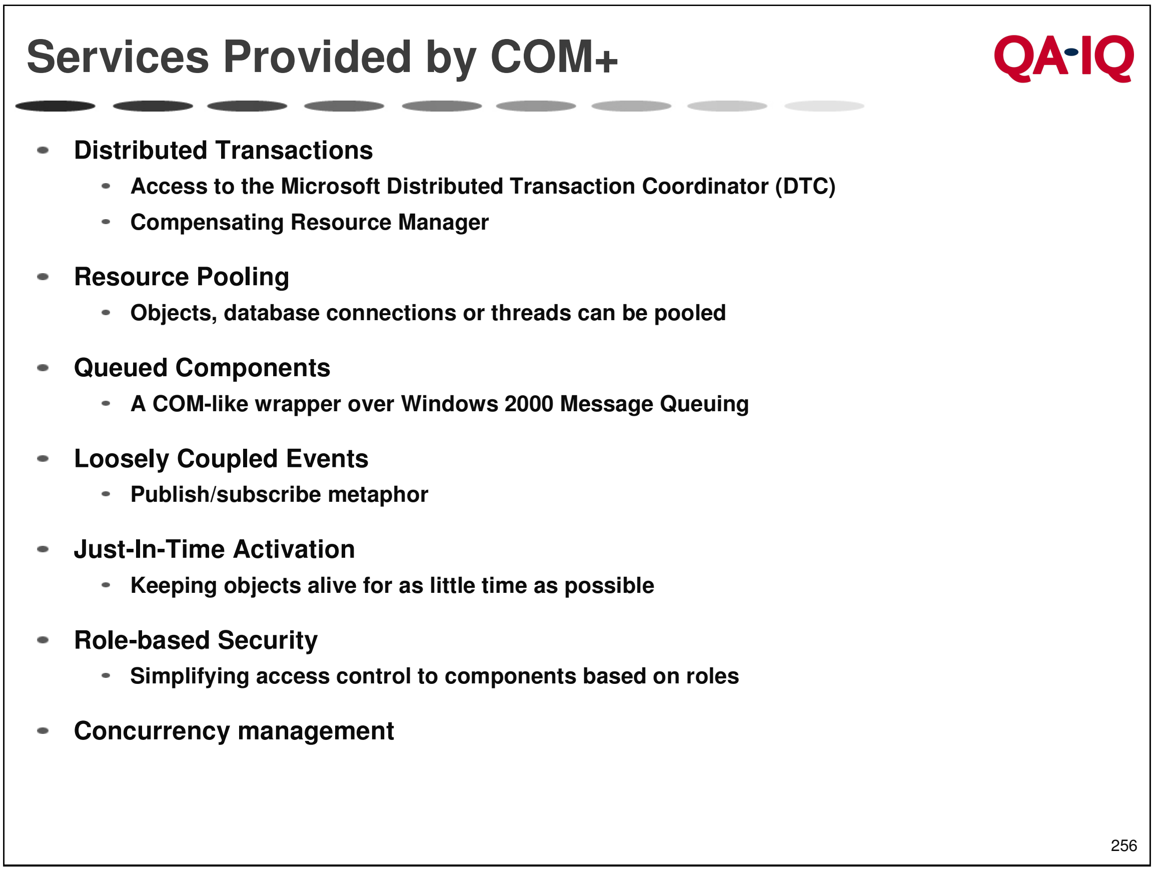
Given the large number of transactions that occur through a web application, these COM components have to be able to be deployed to support both *scalability* and *reliability*. This affects the way that we code components, because they need to be stateless (a single COM component can then support multiple clients), but there should be very little other difference in the way that we work.

COM components also need to be able to enlist in transactions when updating databases, even if they cannot currently gain direct access to the data store directly. This introduces concepts such as transacted message queues, which are provided by MSMQ.

When you add all of these service demands together, it is easy to imagine that they could be provided in a way that makes it easy for the COM developer to gain access to them.

Microsoft introduced, with Windows NT 4, a set of services called MTS. However, this was an add-on to NT, and was divorced completely from COM. Consequently, COM developers had to do core tasks, such as creating objects, in a non-standard way.

With Windows 2000, Microsoft consolidated the MTS and COM systems into a single whole. This enables COM developer to use (or ignore) COM+ services at will - and to code in a standard way no matter which option they select.



Services Provided by COM+

QA-IQ

- Distributed Transactions
    - Access to the Microsoft Distributed Transaction Coordinator (DTC)
    - Compensating Resource Manager
- Resource Pooling
    - Objects, database connections or threads can be pooled
- Queued Components
    - A COM-like wrapper over Windows 2000 Message Queuing
- Loosely Coupled Events
    - Publish/subscribe metaphor
- Just-In-Time Activation
    - Keeping objects alive for as little time as possible
- Role-based Security
    - Simplifying access control to components based on roles
- Concurrency management

256

In enterprise development there are a number of services that are often needed by developers. COM+ goes some way to making them easy to use.

The most obvious requirements for large scale data-driven applications is the need to be able to support transactions. For example, if you are writing a bank component (and you will!), then you would need to ensure that a transfer of funds from one account to another completed correctly.

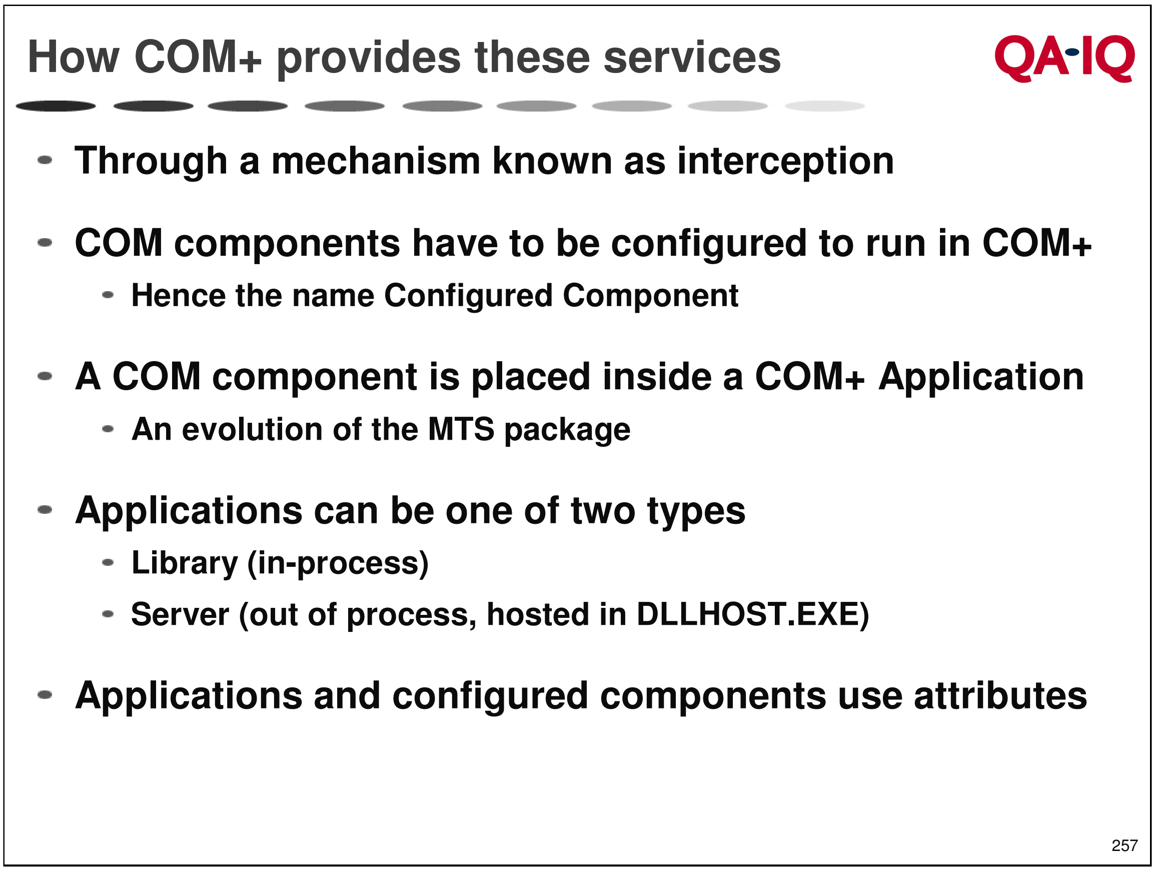
Scalability and reliability place certain demands on systems, designs and developers. COM+ offers facilities such as Just-In-Time Activation (JITA) and Object Pooling to assist in writing scalable systems. With Object Pooling, it is possible to configure a component so that only a defined minimum and maximum number of objects can be in existence at any one time (much like ADO connection pooling).

JITA allows us to perform expensive initialisation code literally just before it is needed, rather than when the object is created. This can be very effective, especially when combined with object pooling.

There are also often cases where your component will not have direct access to the data store, but you still need to be able to ensure that, say, an order will arrive and be processed correctly. Microsoft have provided a transacted queuing system, MSMQ, for a number of years. However, the MSMQ API is not natural for component developers, so with COM+ they have provided an easier mechanism for using it known as Queued Components.

COM+ provides a publisher/subscriber event mechanism, which can deliver events either over straight COM calls or via queued components.

Security is fundamental to the enterprise application, and COM+ offers a simple role-based security mechanism that can be used to control access to applications and configured components. This can greatly simplify security programming, especially if you run the application under a known domain security account.



How COM+ provides these services

QA-IQ

- Through a mechanism known as interception
- COM components have to be configured to run in COM+
  - Hence the name Configured Component
- A COM component is placed inside a COM+ Application
  - An evolution of the MTS package
- Applications can be one of two types
  - Library (in-process)
  - Server (out of process, hosted in DLLHOST.EXE)
- Applications and configured components use attributes

257

So how do we, as COM component developers, avail ourselves of these services?

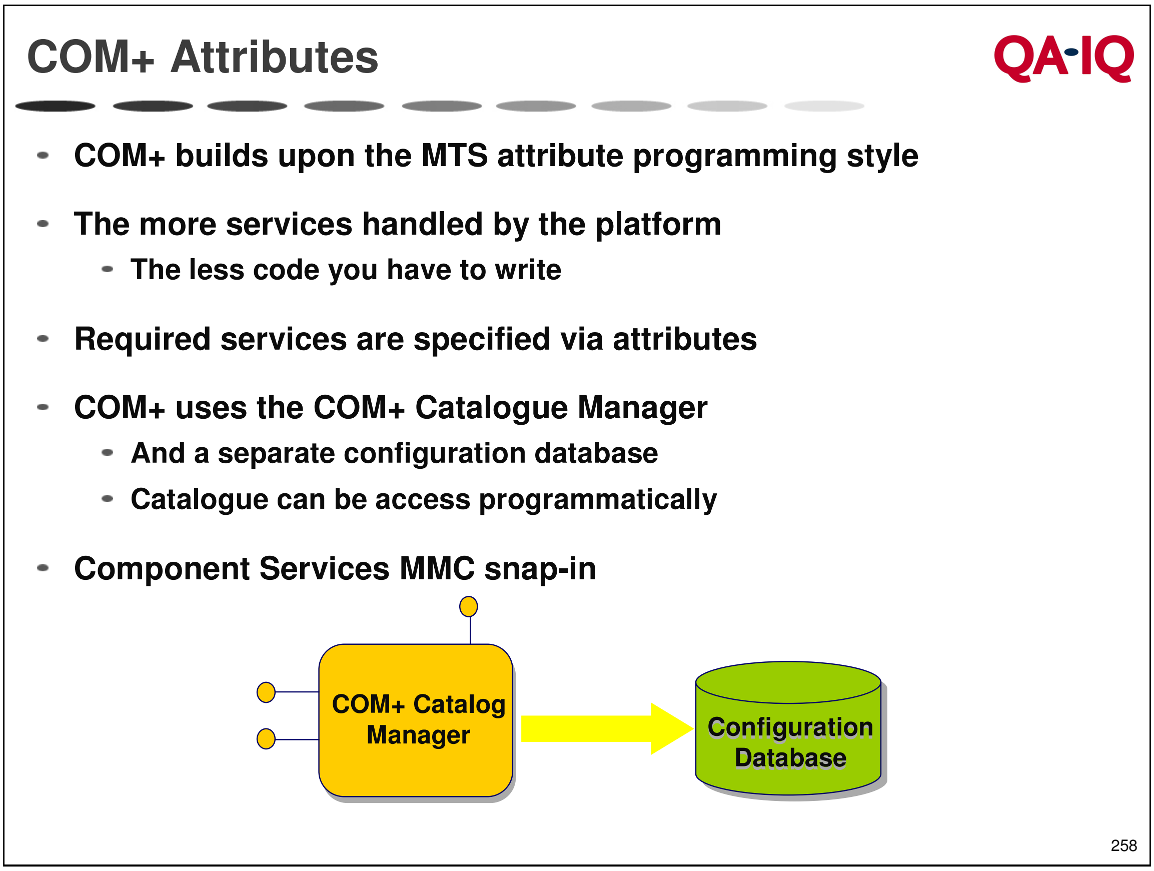
COM+ uses a very simple technique to provide most of these services - interception. In other words, COM+ needs to pre- and post-process the calls through to the methods on the component. We will find out more about interception in a few pages time.

How does COM+ know about the component's requirements? Very simply stated, as a developer of a component that requires these services, you have to register the component with COM+. When a component is registered in this way, it is known as a *Configured Component*.

Configured components live inside a single COM+ Application. This is simply a collection of configured components which will share certain management attributes, such as whether they are going to run in-process or out of process. Note that an application can contain components from multiple COM DLL servers, and that components from within the same server *can* reside in different COM+ applications.

The COM+ runtime (which is fully integrated into the COM runtime) will intercept any creation requests, such as `CoCreateInstance()`, and will determine whether the component is a configured component. If it is, COM+ will ensure that the object is created in an environment that will meet its requirements.

So how does the COM+ configured component provide this information to the COM+ runtime? Through *attributes*.



COM+ Attributes

- COM+ builds upon the MTS attribute programming style
- The more services handled by the platform
  - The less code you have to write
- Required services are specified via attributes
- COM+ uses the COM+ Catalogue Manager
  - And a separate configuration database
  - Catalogue can be access programmatically
- Component Services MMC snap-in

COM+ Catalog Manager
Configuration Database

258

COM components provide information about themselves, such as their type, threading model and so on, to the COM runtime via the registry. COM+ configured components provide additional information as attributes in the COM+ Catalogue.

Remember, the COM runtime will intercept any creation requests and check out whether

(a) the component is a configured component, and

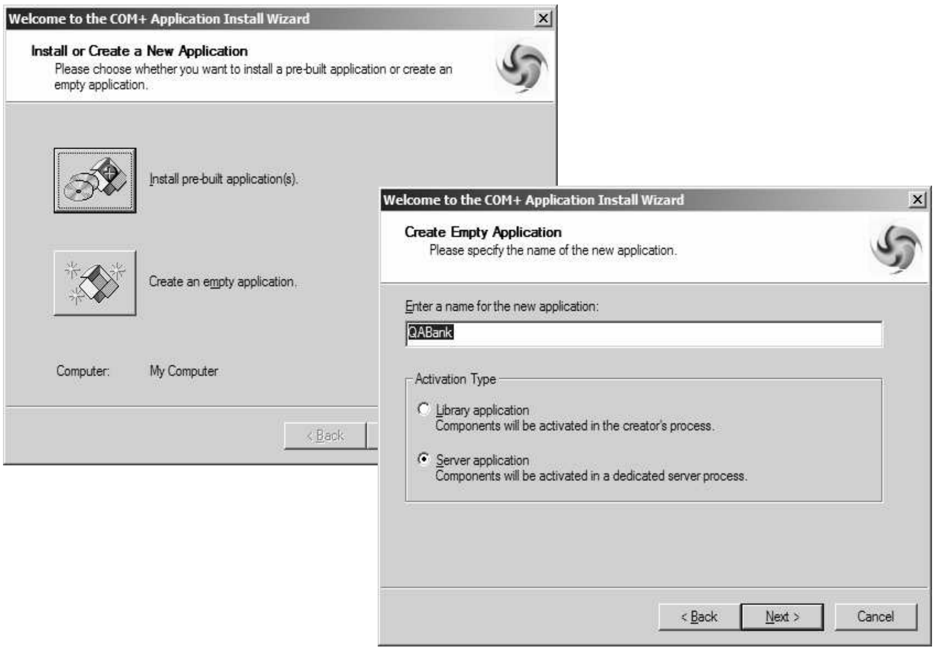
(b) what it's attributes are in the COM+ Catalogue

automatically.

The author of the configured component will establish these attributes either via the Component Services MMC snap-in, or programmatically through the `COMAdminXXX` objects. Note that the use of the programmatic approach is beyond the scope of this course, but suffice to say that it is optimised for use by scripting clients and Visual Basic developers.

So let's take a look at how we create a COM+ application and configure a simple component using the MMC snap-in.

Creating a COM+ Application


Welcome to the COM+ Application Install Wizard
< Back
Welcome to the COM+ Application Install Wizard
< Back
Next >
Cancel

Install or Create a New Application

Please choose whether you want to install a pre-built application or create an empty application.

Install pre-built application(s).

Create an empty application.

Computer: My Computer

Create Empty Application

Please specify the name of the new application.

Enter a name for the new application: QABank

Activation Type

○ Library application
Components will be activated in the creator's process.

● Server application
Components will be activated in a dedicated server process.

259

The first step to creating a configured component is to create a COM+ application.

This is perhaps done most easily using the Component Services MMC snap-in, which can be found under the *Administrative Tools* section of *Control Panel* (or possibly it will be on your *Start Menu, All Programs, Administrative Tools*).

You can expand the tree to see the COM+ Applications. Right-click on *COM+ Applications* and then select *New, Application* to start the COM+ Application Install Wizard.

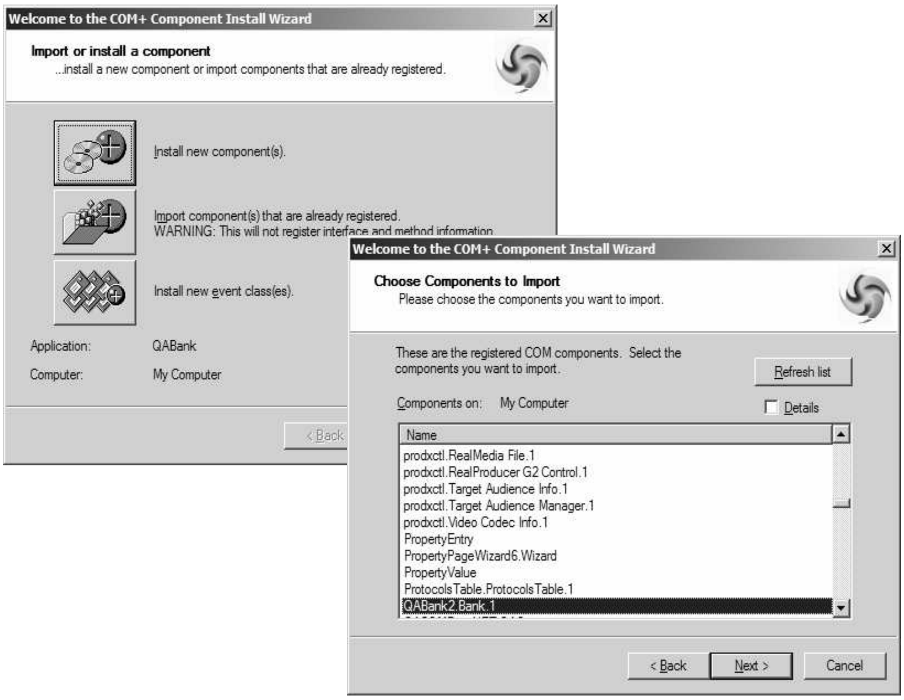
You can either install a pre-built application (one that was written and deployed from another machine), or you can create an empty (new) application. If you take this latter option, you can then specify the name of the application and its type.

You can either have *Library applications* (useful for components that will be used across multiple applications) or *Server applications*. The difference between the two is that configured components from a Library application will run in-process. Components in a Server application will run out-of-process inside a surrogate process called dllhost.exe.

It is important to note that a COM component can only be configured to run in a **single** COM+ application - so if you intend to use a component from multiple applications, you must put it into a Library application if you want to avoid the cost of the inter-process communication involved with Server applications.

Configuring a Component

QA·IQ



Welcome to the COM+ Component Install Wizard

Import or install a component

...install a new component or import components that are already registered.

Install new component(s).

Import component(s) that are already registered.
WARNING: This will not register interface and method information.

Install new event class(es).

Application: QABank

Computer: My Computer

Welcome to the COM+ Component Install Wizard

Choose Components to Import

Please choose the components you want to import.

These are the registered COM components. Select the components you want to import.

Components on: My Computer

□ Details

```
Name
prodxctl.RealMedia File.1
prodxctl.RealProducer G2 Control.1
prodxctl.Target Audience Info.1
prodxctl.Target Audience Manager.1
prodxctl.Video Codec Info.1
PropertyEntry
PropertyPageWizard6.Wizard
PropertyValue
ProtocolsTable.ProtocolsTable.1
QABank2.Bank.1
```

260

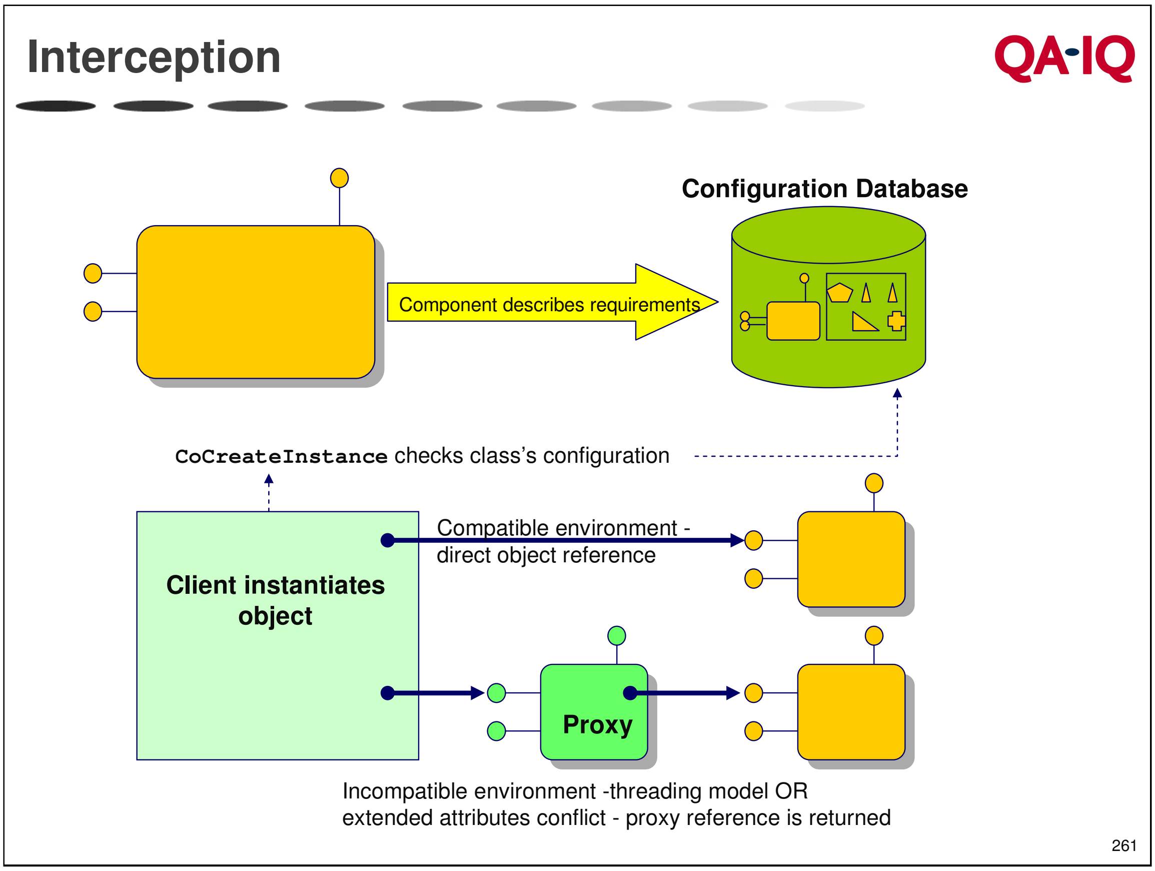
Having created you COM+ application, it is now time to configure your COM component. If you expand your application entry in the Component Services snap-in, you will find a folder called *Components*. Right-click on this entry and choose *New*, *Component*.

You can now add either a whole new component, an existing, registered COM component or a new event class (for Loosely Coupled Events).

You can choose (by Prog ID) a component, and it will instantly become configured. This means that COM+ will store additional information about the component in its catalogue, so that it can intercept any creation requests and ensure the correct environment is established.

WARNING: If you are using custom interfaces, ensure that the proxy/stub DLL is registered for the component's interfaces. If it is not, then all creation requests are going to fail with `E_NOINTERFACE`. If you are using dual or `IDispatch` interfaces, then ensure that the type library is registered.

Congratulations, your component is now configured! Of course, you've yet to describe the services that you want, and how precisely you are going to use them. We'll investigate these properties in the coming pages, but for now let's look at how COM+ provides the interception mechanism that is needed by these services.



Interception

Configuration Database

Component describes requirements

`CoCreateInstance` checks class's configuration

Client instantiates object

Compatible environment - direct object reference

Proxy

Incompatible environment -threading model OR extended attributes conflict - proxy reference is returned

261

So what changes on the client to let them use a configured COM+ component? Absolutely nothing with COM+ (this is and was **NOT** true with MTS - COM+'s precursor).

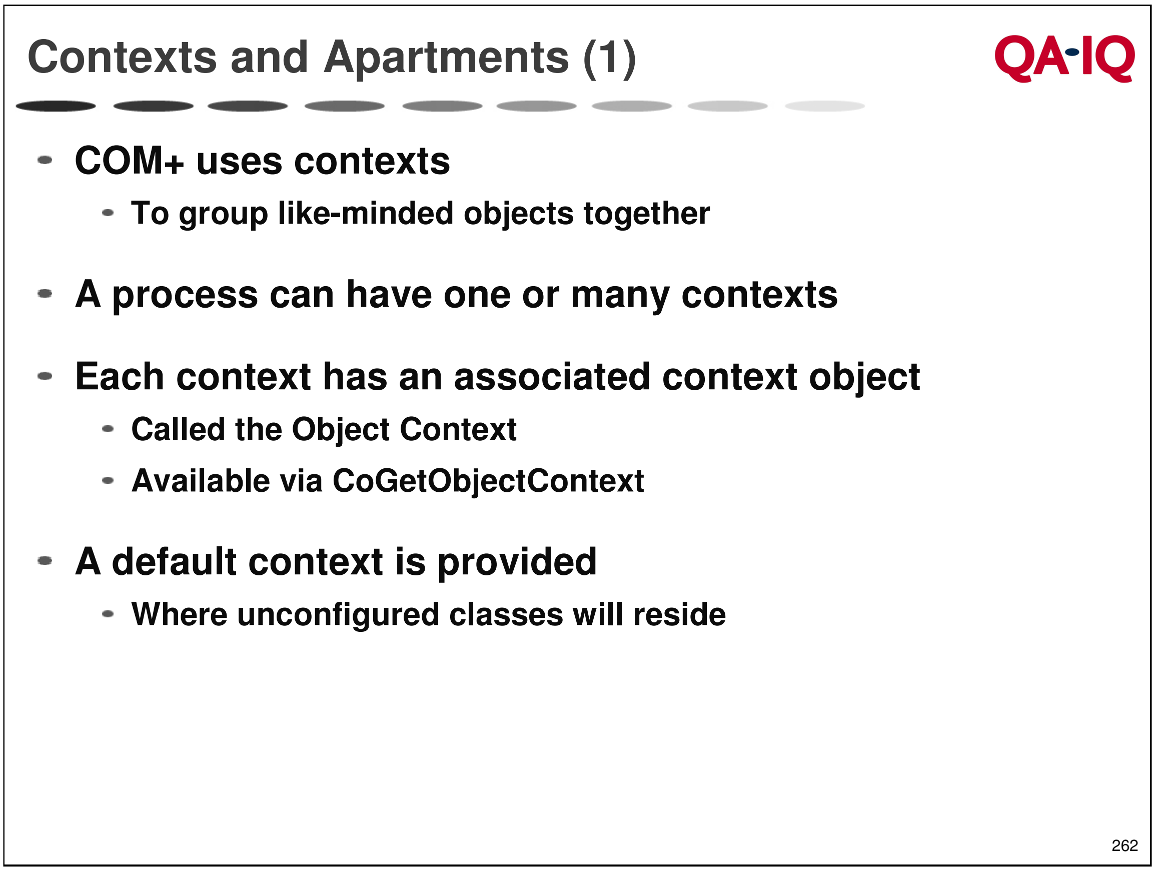
The COM runtime has been augmented with COM+ functionality on Windows 2000 and beyond, which means that it knows to check the COM+ catalogue before instantiating the object.

If a component is registered in the catalogue, the COM+ runtime will validate the client's environment (technically known as a *context*) and will compare the attributes from there to those required by the component. This is very similar to the way that the COM runtime compares apartment requirements in classic COM.

If the COM+ runtime determines that the object and the client have compatible requirements, the object will be placed in the caller's context, and a direct reference will be provided.

If the client and the object have different requirements, a proxy is inserted between the client and the object, and the object will be placed into a compatible context (very probably its own context).

So what's a context, and how do they fit in with apartments?



Contexts and Apartments (1)

QA-IQ

- COM+ uses contexts
  - To group like-minded objects together
- A process can have one or many contexts
- Each context has an associated context object
  - Called the Object Context
  - Available via CoGetObjectContext
- A default context is provided
  - Where unconfigured classes will reside

262

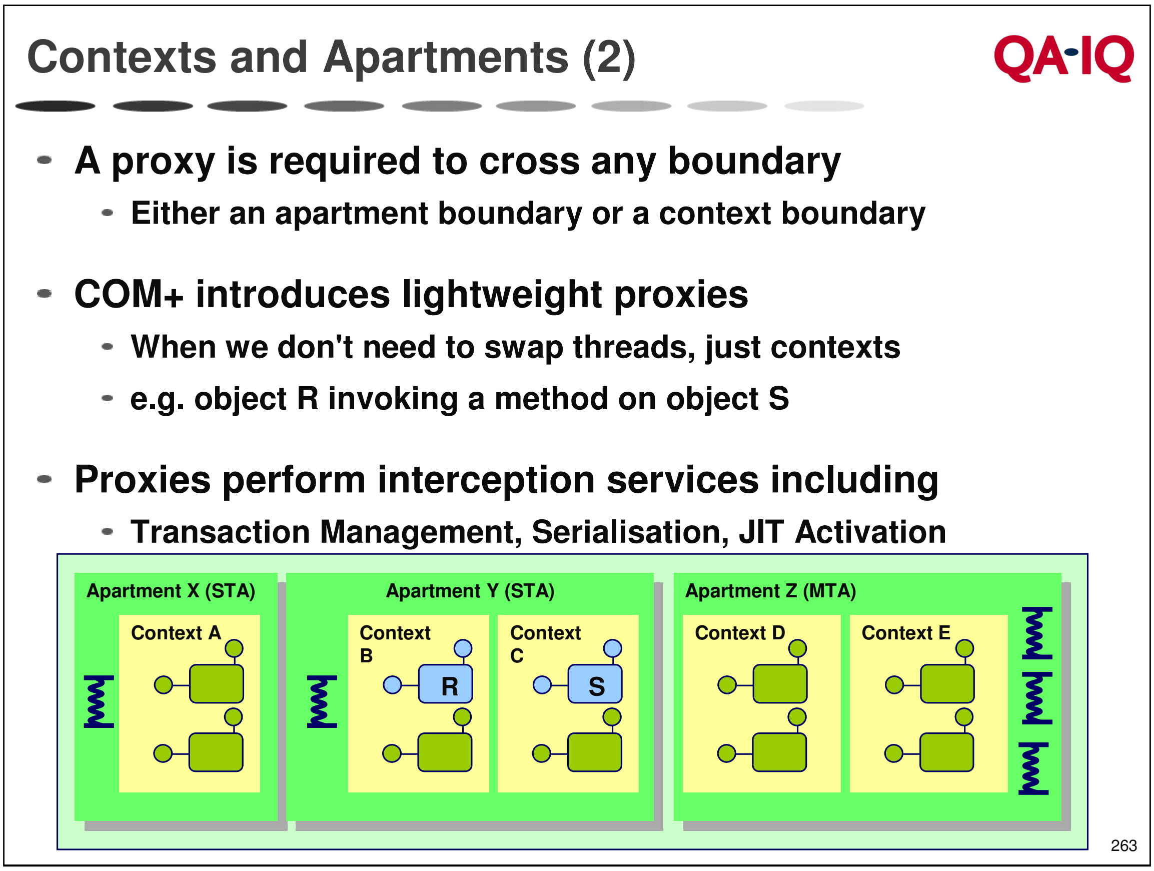
A context is simply a way of grouping objects with compatible service requirements together, much like an apartment is used to group objects with certain threading requirements.

There are multiple contexts within a process (in many cases, each individual object will receive its own context). We can identify the context for an object through a new API call, `CoGetObjectContext()`.

Using this API, we can determine information about the context, such as whether we are in a transaction or not, the transaction ID, etc. This also provides us with a mechanism that we can use to commit or abort transactions, as we'll see later on.

Contexts occur in all COM and COM+ programming, so what happens to unconfigured (legacy) COM components? Within each apartment is a default context, into which they are placed. Obviously, these legacy components cannot partake of any COM+ services, most of which are provided through interception and are based on information held in the context object.

Note: There is another form of context, known as *call context*, which applies to a logical sequence of calls and not to a single object. We can use call context to find out information about the security principals that have been established when the call was made.



Contexts and Apartments (2)

- A proxy is required to cross any boundary
  - Either an apartment boundary or a context boundary
- COM+ introduces lightweight proxies
  - When we don't need to swap threads, just contexts
  - e.g. object R invoking a method on object S
- Proxies perform interception services including
  - Transaction Management, Serialisation, JIT Activation

Apartments are still alive and well in COM+, and behave in exactly the same way that they did in classic COM. However, there is the concept of this element of finer granularity, the context, in COM+ that is used to provide access to some of COM+'s services.

Contexts reside in one and only one apartment. Multiple contexts can live in a single apartment, as is shown in the diagram above. Normal apartment rules are obeyed for which thread is used to dispatch the calls to the COM objects in the apartments.

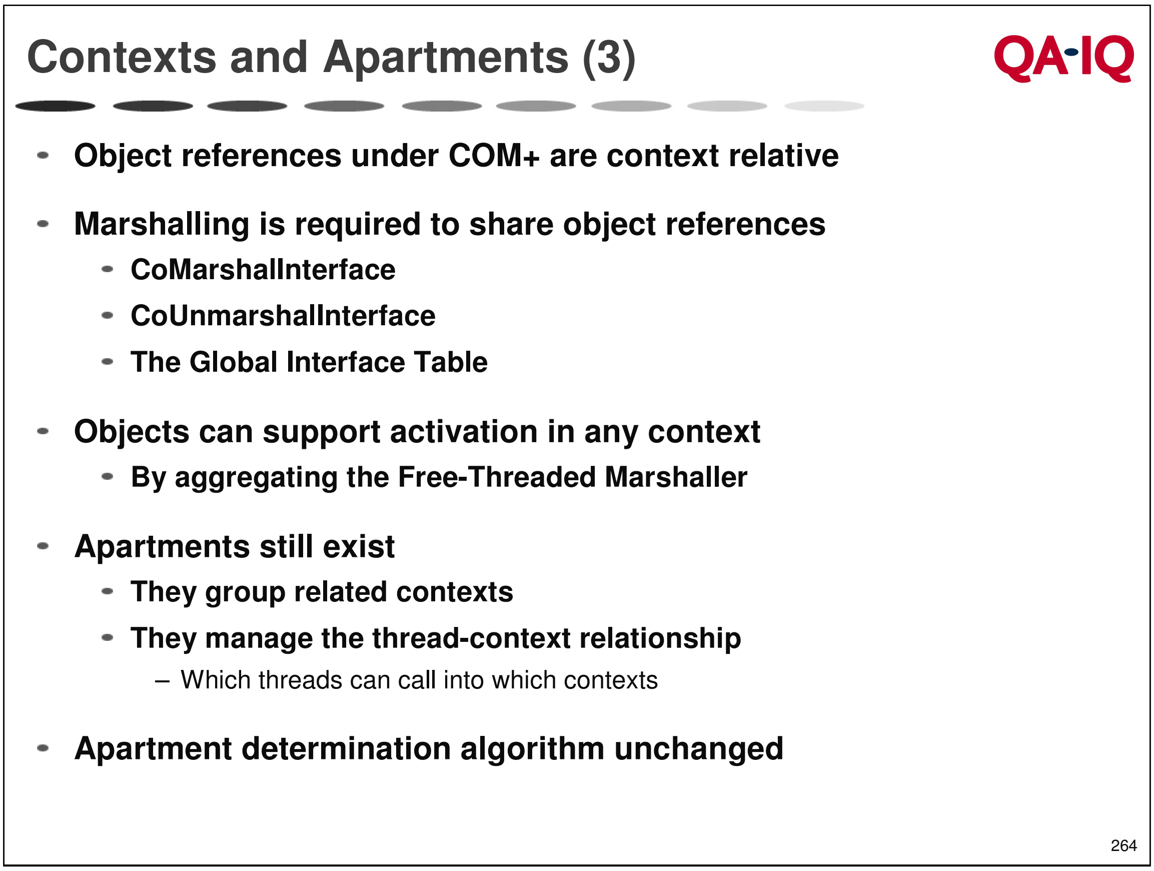
However, there is one extra level of containment, as shown. What this means is that in COM+, even crossing a *context* boundary demands the use of a proxy. This sounds painful and slow (it can be - design and deploy with context in mind), so COM+ introduced the concept of the lightweight proxy.

A lightweight proxy is a proxy that does NOT require a thread switch. For example, if object R was making a method call on object S, the same thread would be used, but the context would need to be set so that object S received its context information.

It should be noted that many COM+ services require the use of proxies to perform pre- and post-method processing, but not all. Object pooling, for example, does not require any specific context information, so a pooled object can *potentially* run inside its caller's context.

The COM+ component configuration tool lets you specify that a component must be instantiated in the caller's context, by setting the *Must be activated in caller's context* flag in the *Activation* tab of the configured component's properties dialog.

If the COM+ runtime detects an incompatibility between the caller's and the object's context, then the creation call will fail with the well known `HRESULT` `CO_E_ATTEMPT_TO_CREATE_OUTSIDE_CLIENT_CONTEXT`.



Contexts and Apartments (3)

QA-IQ

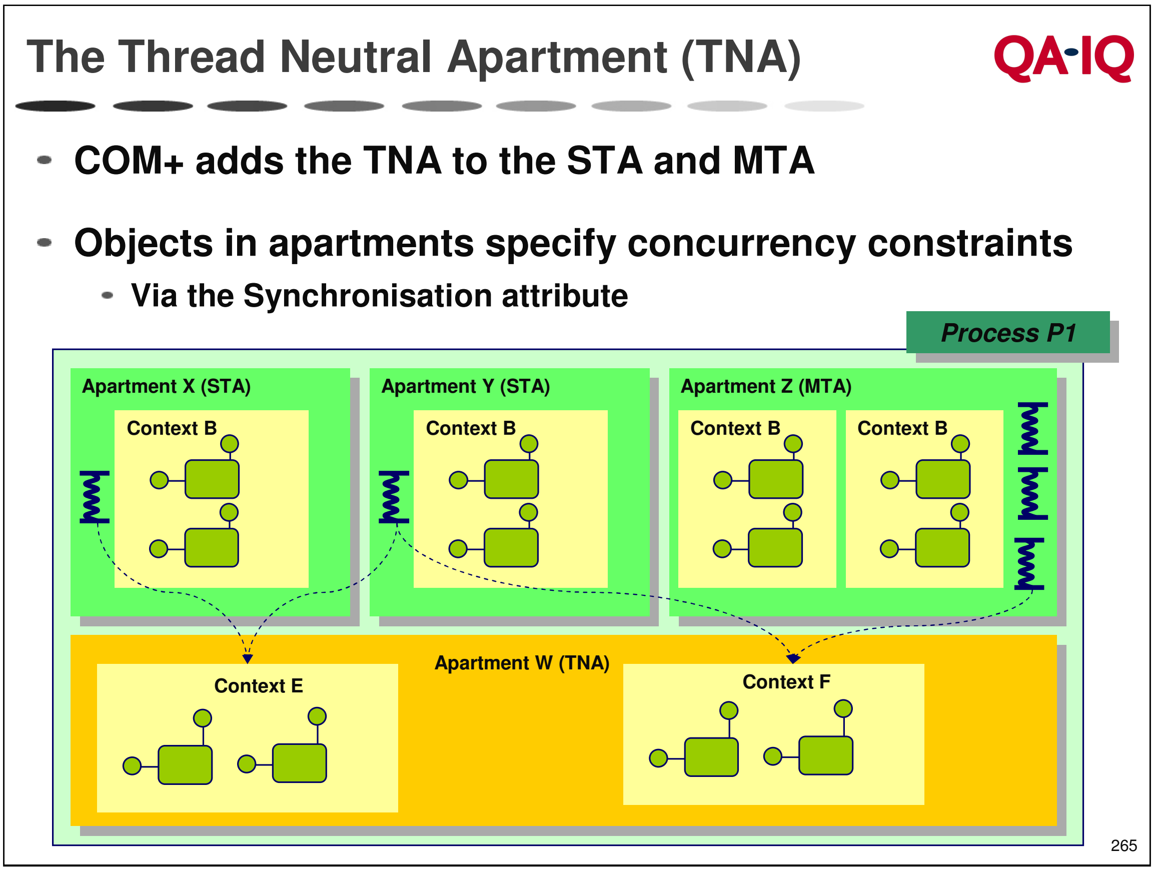
- Object references under COM+ are context relative
- Marshalling is required to share object references
  - CoMarshalInterface
  - CoUnmarshalInterface
  - The Global Interface Table
- Objects can support activation in any context
  - By aggregating the Free-Threaded Marshaller
- Apartments still exist
  - They group related contexts
  - They manage the thread-context relationship
    - Which threads can call into which contexts
- Apartment determination algorithm unchanged

264

Object references (Ixxx *'s) are context relative in COM+. This means that if you hold a reference in one context, it will have to be marshalled to another context.

The functions `CoMarshalInterThreadInterfaceInStream()` and `CoGetInterfaceAndReleaseStream()` can be used to marshal object references across contexts, as with the arrival of COM+ these APIs became context aware.

An object can be created in ANY context so long as it is compatible with the client's context, and it has a compatible threading model (remember, a context is contained within a single apartment). But what happens if it isn't, and yet you don't want the cross context overhead? All you have to do is understand the rules of the Free Threaded Marshaller (FTM) and aggregate it. This is a fairly advanced subject that is really beyond the scope of this course. Just remember with any object that aggregates the FTM that any object references you store will probably be context relative, so you will need to store them in the Global Interface Table, and extract them as you need them.



The Thread Neutral Apartment (TNA)

QA-IQ

- COM+ adds the TNA to the STA and MTA
- Objects in apartments specify concurrency constraints
  - Via the Synchronisation attribute

Process P1

Apartment X (STA)

Context B

Apartment Y (STA)

Context B

Apartment Z (MTA)

Context B

Context B

Apartment W (TNA)

Context E

Context F

Windows 2000 saw the arrival of a new apartment type, the Thread Neutral Apartment.

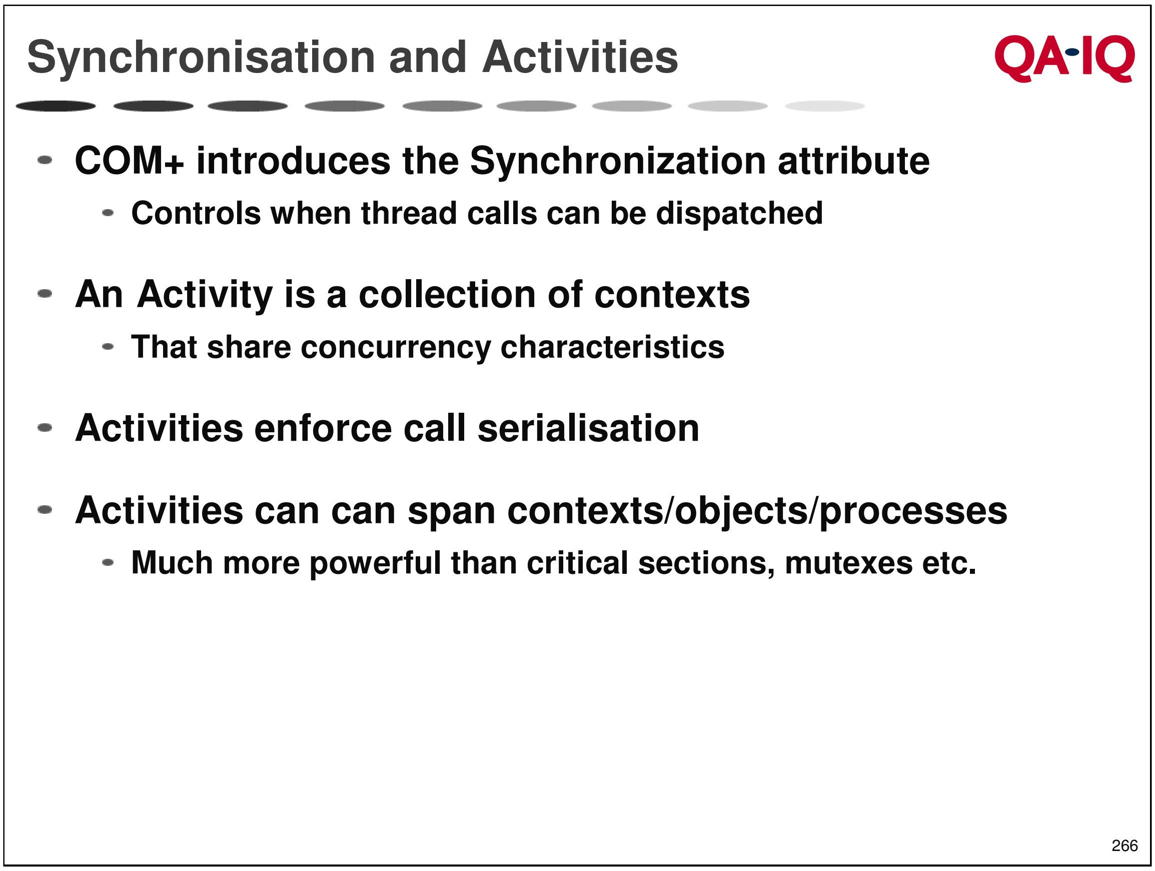
The TNA contains no threads of its own. Instead, when a method is invoked on an object that lives in the TNA, a lightweight proxy is used to swap the calling thread's context to that which is containing the object.

Objects that want to live in the TNA can indicate this preference by setting their `ThreadingModel` registry entry to "Neutral". Also note that a COM object that sets its `ThreadingModel` to "Both" is also indicating that it can live in a TNA (for Both you should read ANY)!

Given that calls into the TNA can be executed on any type of thread, you must be conscious of the restrictions that apply to all apartment types (don't block an STA thread and you won't have thread affinity).

Is it possible to specify that the object should live in the TNA to avoid the full proxy thread swapping overhead, but that you still want the object to be protected against concurrent access? Absolutely!

When configuring your component, you can set the *Synchronisation* attribute (on the component's *Concurrency* tab in the properties dialog) to *Required*.



Synchronisation and Activities

- COM+ introduces the Synchronization attribute
  - Controls when thread calls can be dispatched
- An Activity is a collection of contexts
  - That share concurrency characteristics
- Activities enforce call serialisation
- Activities can can span contexts/objects/processes
  - Much more powerful than critical sections, mutexes etc.

COM+ handles concurrency and synchronisation much better than classic COM. COM+ introduces the concept of an *Activity* which is used to represent a *logical* thread of execution as calls are made from object to object.

You can imagine an activity as offering similar functionality to an STA, in that it represents a synchronisation boundary, but it can span multiple contexts, apartments and even processes and machines. Even though calls may be being made using different physical threads, any attempt to access a component concurrently that is already running in an activity will be prevented.

Whilst activities prevent concurrent access, there is often a need for re-entrant access to a component within an activity. COM+ handles this using a *Causality ID* which is passed from context to context as calls are made. For example, if A calls B and then B calls C, A is said to be causally related to C (they are part of the same logical call stack).

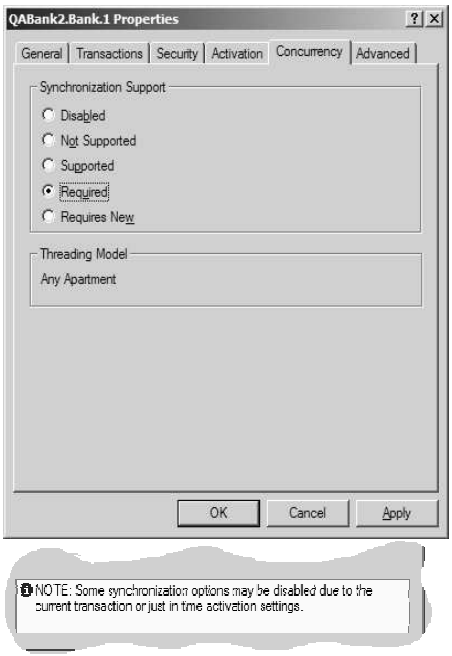
Consequently, when calling into an activity, COM+ will check the causality ID of the incoming call. If the causality matches that which is established for the activity, COM+ knows that we can safely re-enter an activity that is already part of the call chain. COM+ will block any incoming call to an activity which has a different causality ID.

This mechanism prevents deadlock whilst offering synchronisation.

All of this is controlled by the value of the `Synchronisation` attribute that we saw on the previous slide.

Configuring Concurrent Access

QA-IQ



Synchronization Support

○ Disabled
○ Not Supported
○ Supported
● Required
○ Requires New

Threading Model

Any Apartment

NOTE: Some synchronization options may be disabled due to the current transaction or just in time activation settings.

267

Activities flow from context to context as method calls are made, and the synchronisation setting determines whether it will run in an activity or not.

So what do the various settings mean?

Disabled means that the component really knows nothing about activities, and is useful for legacy COM components that have been migrated to COM+. The synchronisation attribute will be ignored when the object is created, so it may (or may not) share it's caller's context.

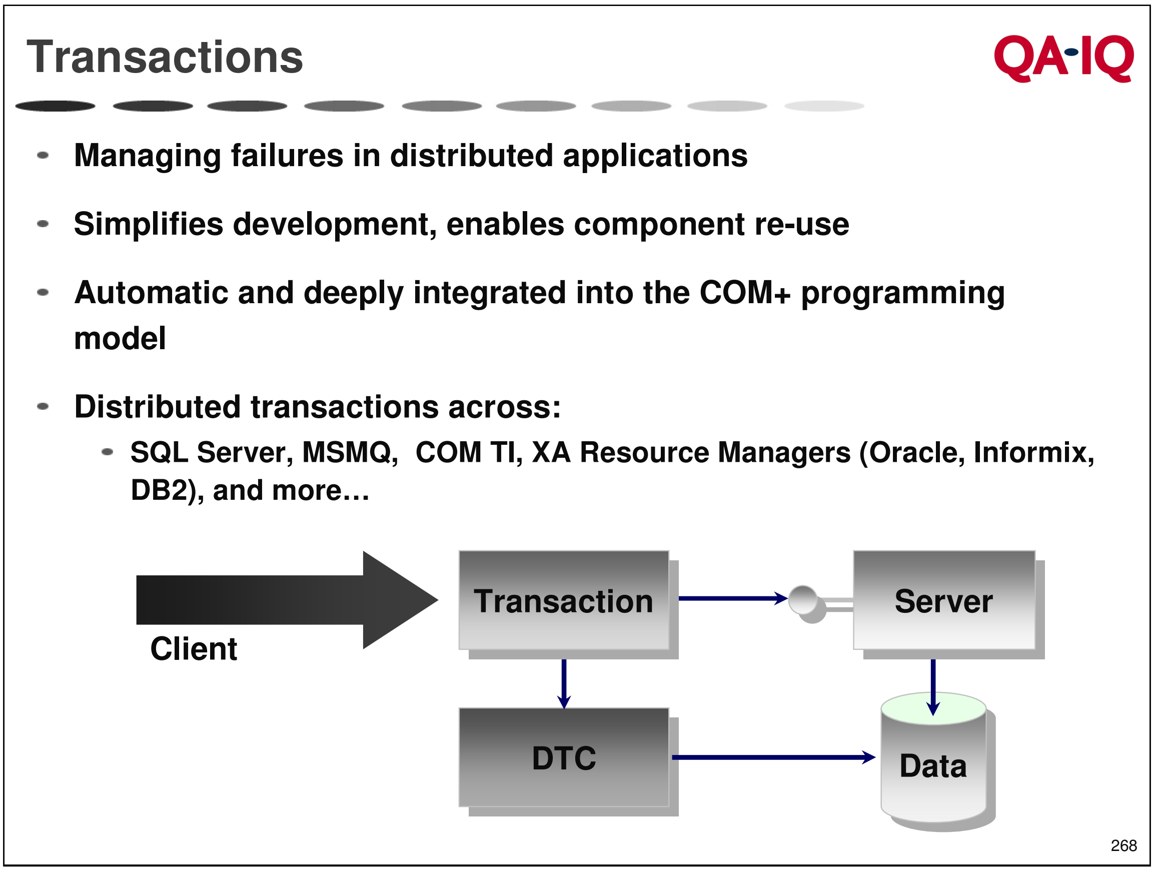
Not Supported means exactly that. This COM+ object will **never** run inside an Activity, and will thus not be synchronised unless it is living inside an STA.

Supported means that the object will happily live inside an activity if there is one around, an will happily run outside of an activity as well.

Required means that the component needs an activity. If there is not one present, then a new one will be created for it. In essence, this object is always protected against concurrent access.

Requires New ensures that this component will always receive its own, new activity.

It should be noted that the settings for this option are often restricted by other settings, such as the JIT Enabled setting or the Transaction setting, as these services mandate certain synchronisation requirements themselves.



Transactions

QA-IQ

- Managing failures in distributed applications
- Simplifies development, enables component re-use
- Automatic and deeply integrated into the COM+ programming model
- Distributed transactions across:
  - SQL Server, MSMQ, COM TI, XA Resource Managers (Oracle, Informix, DB2), and more…

Transactions are critical to enterprise database applications, and COM+ components need to be able to take part in the voting.

It is beyond the scope of this course to explain transactions in detail, but suffice to say that a transaction allows us to guarantee that a series of operations will either all complete or all fail.

Many database engines, such as SQL Server, have the ability to perform *local* transactions. In other words, you can perform a sequence of T-SQL statements on SQL Server under transaction control. But what happens if you need to perform operations across multiple resources, such as a DB2 database, a SQL Server database, drop a message in MSMQ and perform a simple file operation*?

To do this, you need to use a distributed transaction, which is provided for us with Windows 2000 and beyond via the Microsoft Distributed Transaction Coordinator (DTC). Using the DTC, we can run operations on multiple resource managers under a single transaction, check to make sure that everyone is happy, and then commit or abort all of the changes.

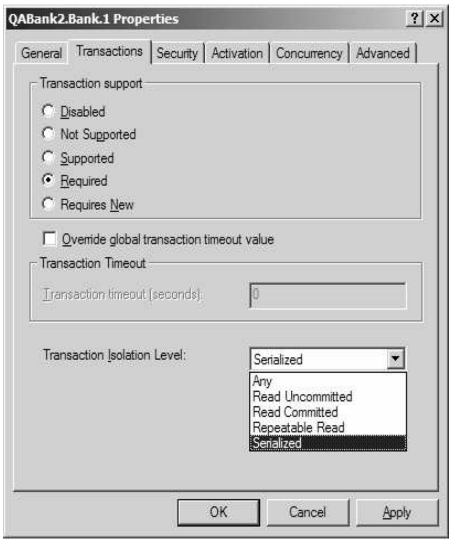
Accessing the DTC using raw programming would be hard and tedious, but as component developers we need our components running under transactions. Microsoft originally offered MTS to help host our COM components, but this has been superseded on Windows 2000 and beyond with COM+!

It should be noted that the Microsoft DTC is a transaction manager that is compatible with any XA-compliant resource manager, so it can be used with many different systems.

* Note that you might need to avail yourself of the services of the Compensating Resource Manager to add transaction support for file operations.

Configuring Transaction Support

QA-IQ



Transaction support

○ Disabled
○ Not Supported
○ Supported
● Required
○ Requires New

□ Override global transaction timeout value

Transaction Timeout

Transaction timeout (seconds): 0

Transaction Isolation Level: Serialized

Any
Read Uncommitted
Read Committed
Repeatable Read
Serialized

269

Again, we configure the transaction attributes using the Component Services snap-in (or programmatically if you prefer).

The settings are very similar to those of the synchronisation attribute, and have the following meaning:

Disabled means that the component has no knowledge of transactions, and consequently COM+ will ignore the transaction setting when determining whether the component is compatible with the current context.

Not Supported means that the component does not support transactions, and the object will only be created in the caller's context if the caller also has this *Not Supported* setting.

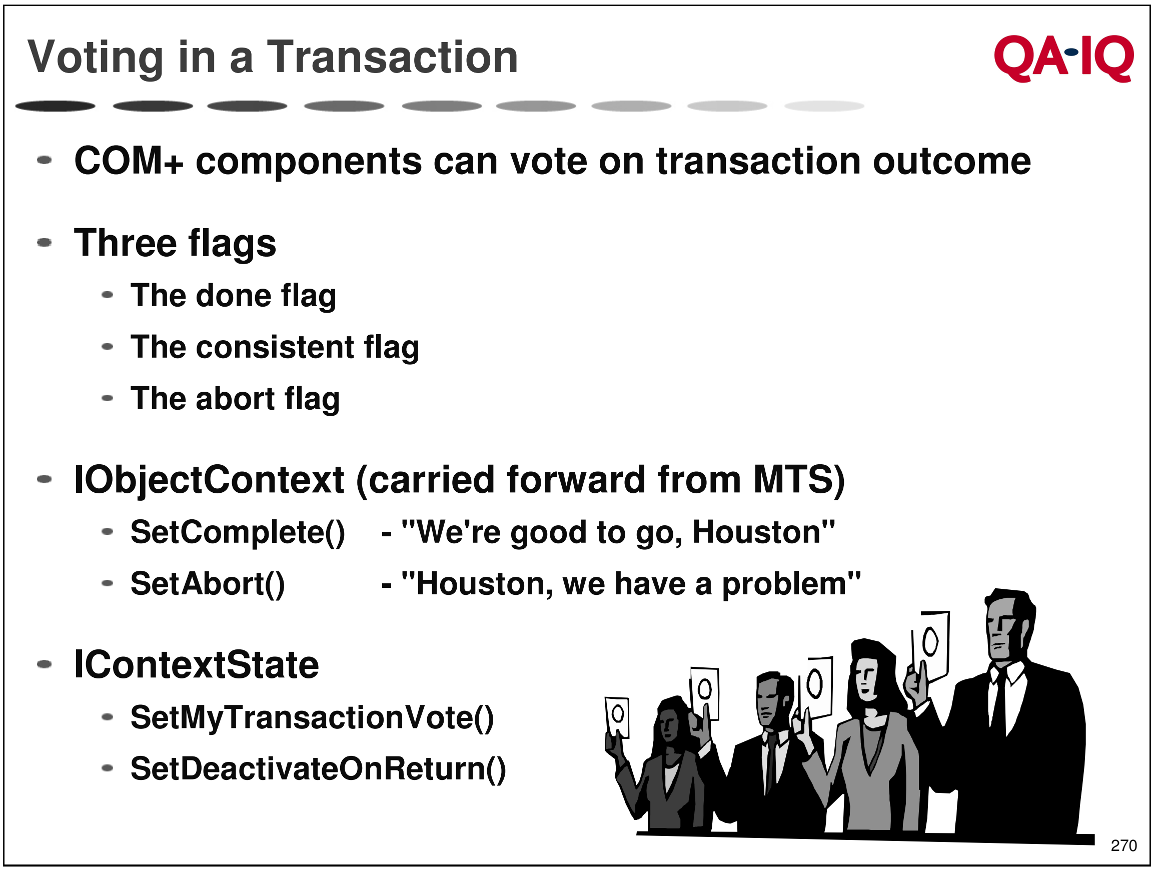
Supported means that the component will run inside the caller's transaction if one is present, otherwise it will run happily without a transaction.

Required means that a transaction is needed for this component to run. If a transaction is present, then the component will use that transaction, otherwise a new one will be created.

Requires New means that COM+ will always create a new transaction for this component. If this component votes to abort the transaction, it will not affect the caller's transaction.

With transactions, we also introduce the concept of the *transaction isolation level* (the *I* in *ACID*). *Dirty reads* occur when you read data that has been changed by another transaction that has yet to commit, and *non-repeatable reads* can happen when you read data twice within a transaction, but it has been changed between the reads. Also, you might read data that might not really exist in the database (*phantoms*), which have been added or are deleted by other users while your transaction is running.

You can adjust the isolation level to overcome these problems, with *Serializable* the easiest to code for the developer, as none of the above situations should arise. Be aware that this has the greatest performance impact on the system.



Voting in a Transaction

QA-IQ

- COM+ components can vote on transaction outcome
- Three flags
  - The done flag
  - The consistent flag
  - The abort flag
- IObjectContext (carried forward from MTS)
  - SetComplete() - "We're good to go, Houston"
  - SetAbort() - "Houston, we have a problem"
- IContextState
  - SetMyTransactionVote()
  - SetDeactivateOnReturn()

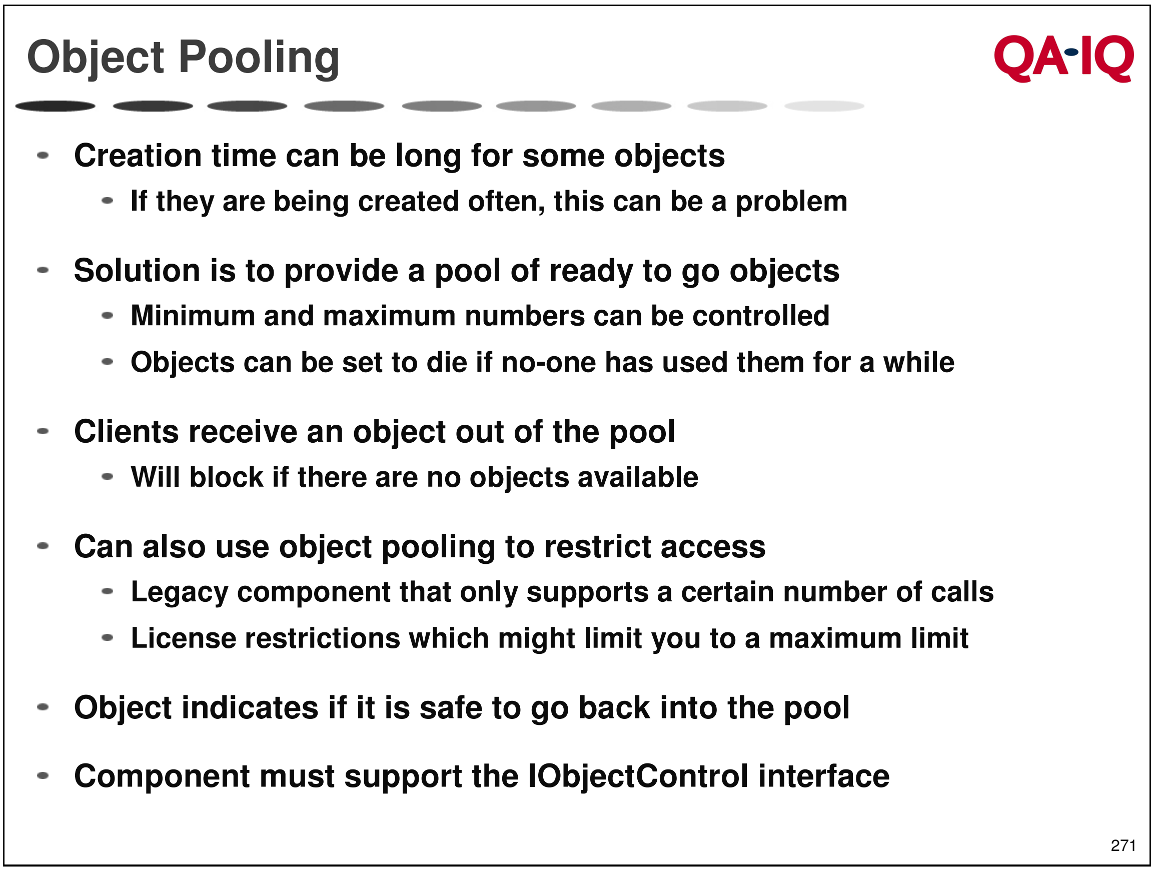
270

COM+ components need to be able to vote on whether a transaction can commit or not. COM+ maintains three flags to assist with this voting process.

The done flag indicates that the component has finished all of its processing or not. If this is true, the object can be destroyed or pooled. When this flag is set to true, no more information on the transaction can be determined other than that contained in the consistent flag

The consistent flag indicates whether the transaction can be committed or not. If this flag is set to false, then the transaction cannot commit. Note that if this flag is set to false and the done flag are set to true, no further calls are allowed into the context and the transaction is doomed. In this case, COM+ will set the abort flag to true.

Setting these flags is done either using the legacy interface `IObjectContext`, which sets both flags at once, or using the more modern `IContextState` interface that allows us to set each flag.



Object Pooling

- Creation time can be long for some objects
  - If they are being created often, this can be a problem
- Solution is to provide a pool of ready to go objects
  - Minimum and maximum numbers can be controlled
  - Objects can be set to die if no-one has used them for a while
- Clients receive an object out of the pool
  - Will block if there are no objects available
- Can also use object pooling to restrict access
  - Legacy component that only supports a certain number of calls
  - License restrictions which might limit you to a maximum limit
- Object indicates if it is safe to go back into the pool
- Component must support the IObjectControl interface

There are some components that can require a large start up time. For example, maybe they are extracting data from databases or from sources over the Internet.

It might be more efficient to maintain a pool of ready built objects that can simply be given out to clients in response to a creation request. This is possible in COM+ through *Object Pooling*.

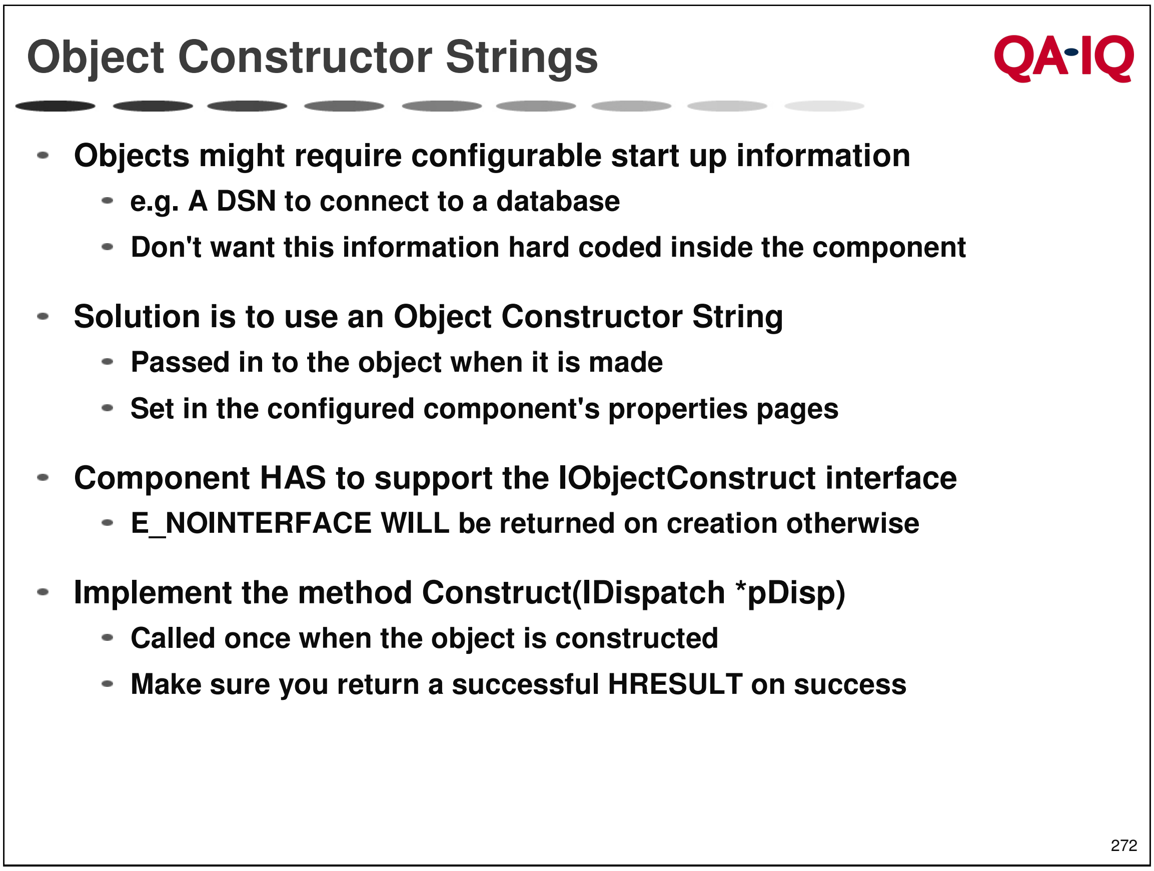
The component needs to implement the `IObjectControl` interface in order to support object pooling, and it has to return `S_OK` from the `CanBePooled()` method to indicate to COM+ that it is OK to put the object back into the pool, as opposed to destroy the object.

Object pooling can also be used to control access to a resource for which you have a restricted number of licenses. You can configure a COM+ component to have an allowed maximum number of instances. This, of course, can also be used to restrict the amount of resources consumed by the objects. When the maximum number of objects is reached, clients will block and wait for an object to become available, up to a certain timeout period.

As well as a maximum, you can also specify a minimum number of objects, so that there are always some initialised and ready to go.

The component can also be configured so that objects die after a certain period of time if they haven't been used. Therefore, in quiet periods (as far as this particular component is concerned), the object count will gradually reduce down to the minimum number.

Be aware that whilst it is possible to pool an object that requires transaction support, additional coding must be performed in the `Activate()` method of `IObjectControl`, so that the component can enlist in the transaction for this particular client.



Object Constructor Strings

- Objects might require configurable start up information
  - e.g. A DSN to connect to a database
  - Don't want this information hard coded inside the component
- Solution is to use an Object Constructor String
  - Passed in to the object when it is made
  - Set in the configured component's properties pages
- Component HAS to support the IObjectConstruct interface
  - E_NOINTERFACE WILL be returned on creation otherwise
- Implement the method Construct(IDispatch *pDisp)
  - Called once when the object is constructed
  - Make sure you return a successful HRESULT on success

272

It is almost a certainty that, at some point in time when developing COM+ components, you will want to provide the object with some information as it is created. This might be a connection string to a database or a location for writing out log information.

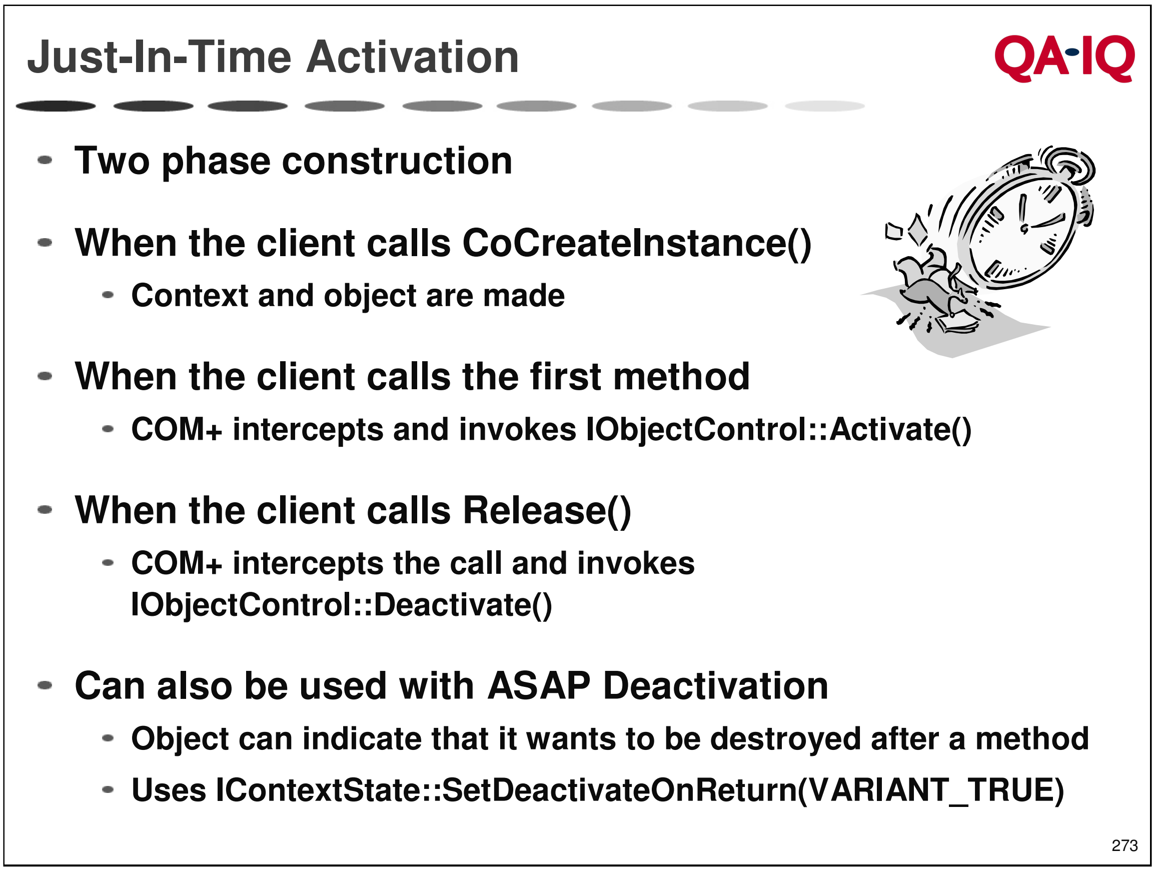
Rather than hard code this information in the component, or extract it from INI files or the registry, we can get COM+ to pass the information to us in an *object constructor string*.

This, as its name suggests, is a simple string that we have to parse ourselves (so, for example, it could be the name of a file with additional information in it!).

In order to support constructor strings, we must configure them in the COM+ Catalogue for this component (typically done by an administrator using the Component Services snap-in). We must also implement the interface `IObjectConstruct` for the COM object.

Note: If you enable an object constructor string in the COM+ catalogue, but fail to implement `IObjectConstruct`, then no instances of your object will EVER be constructed, and the client will receive the well known `HRESULT E_NOINTERFACE`. You have been warned!

When constructor strings are in play, COM+ will call the method Construct() once when the object is created. Unfortunately, it passes you an `IDispatch` object reference (ouch!). However, you can use `QueryInterface()` on this to get an `IObjectConstructString` interface, which has a method `get_ConstructString( [out] BSTR *theString )` which you can use to get the constructor string.



Just-In-Time Activation

QA-IQ

- Two phase construction
- When the client calls CoCreateInstance()
  - Context and object are made
- When the client calls the first method
  - COM+ intercepts and invokes IObjectControl::Activate()
- When the client calls Release()
  - COM+ intercepts the call and invokes IObjectControl::Deactivate()
- Can also be used with ASAP Deactivation
  - Object can indicate that it wants to be destroyed after a method
  - Uses IContextState::SetDeactivateOnReturn(VARIANT_TRUE)

273

JITA provides a mechanism for delaying the expensive creation phase of an object until it is really needed.

Instead of performing initialisation in the C++ constructor (or `FinalConstruct()` if you are using ATL), COM+ will wait until a method is invoked on the object. At that point, COM+ intercepts the call, and invokes the `Activate()` method from `IObjectControl` on your object. You therefore perform your object initialisation here.

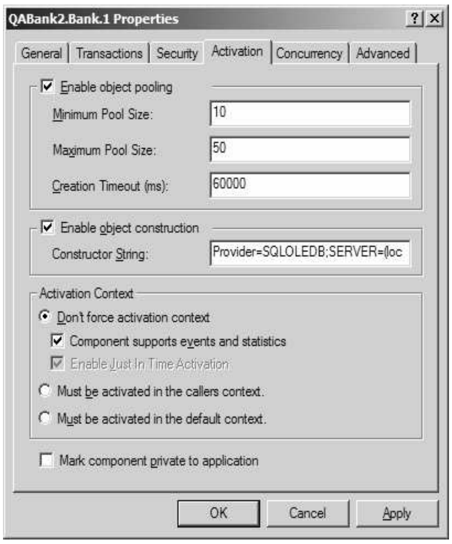
When the client has finished with the object, and calls `Release()`, again COM+ intercepts the call and invoked the `Deactivate()` method.

This has an extra advantage when using *ASAP Deactivation*. This technique lets you indicate from within a method that the object can be destroyed, by setting the "done" bit. COM+ will then call `Deactivate()` and then destroy the object, *even though the client is still holding a reference*. If the client then makes a subsequent method call, COM+ will use JITA to instantiate a new object, and will call its `Activate()` method.

The idea behind JITA and ASAP Deactivation is that the objects will consume less resources, but it only has real advantage if the cost of creation is not too high, and the number of objects being created is not too large.

Note that JITA can be used with pooled objects, and is required for components that support transactions.

Configuring Activation Properties



☑ Enable object pooling

Minimum Pool Size: 10

Maximum Pool Size: 50

Creation Timeout (ms): 60000

☑ Enable object construction

Constructor String: Provider=SQLOLEDB;SERVER=(loc

Activation Context

● Don't force activation context

☑ Component supports events and statistics

☑ Enable Just In Time Activation

○ Must be activated in the callers context.

○ Must be activated in the default context.

□ Mark component private to application

274

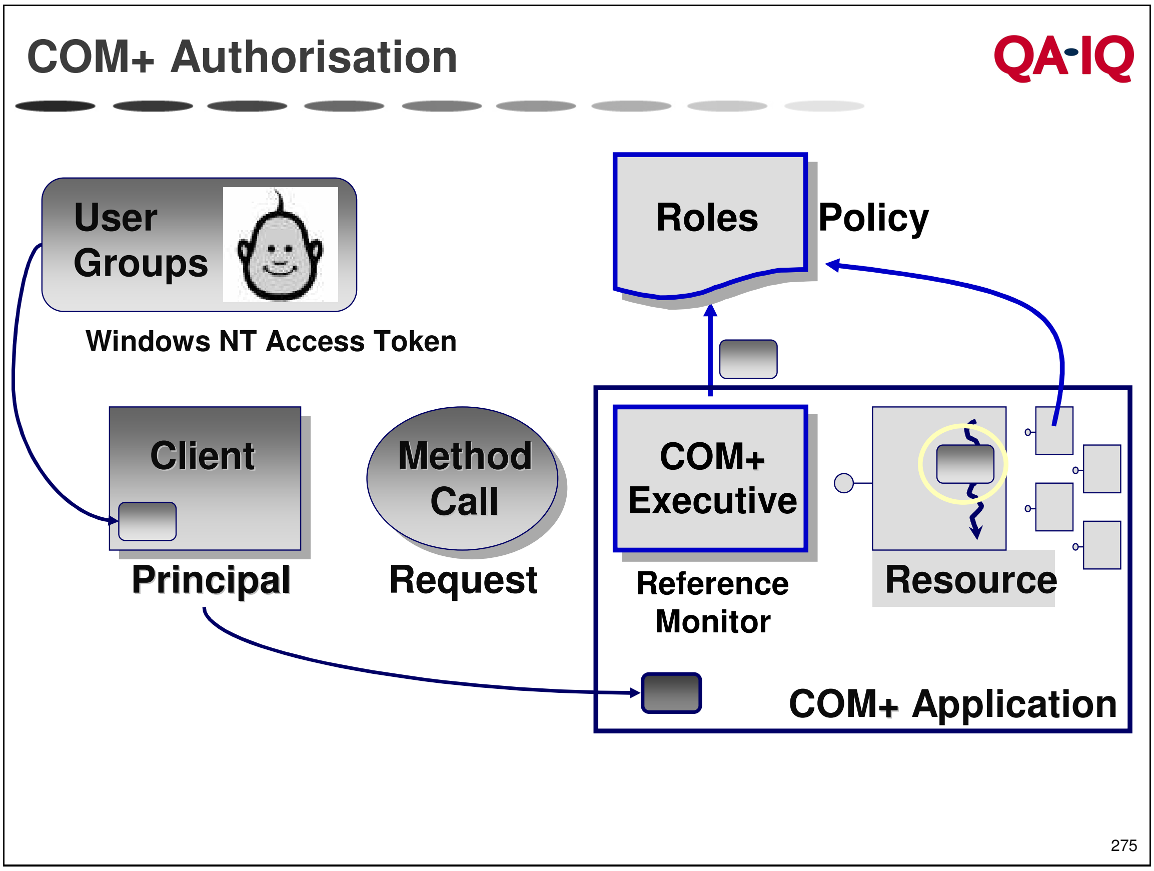
Here we see the settings that control an object's activation in the world of COM+, on the component's *Activation* tab.

Our example above shows an object that will be pooled (with a minimum of 10 and a maximum of 50), that takes a DSN as a constructor string, and which has JITA enabled. Note that JITA is enabled, but that the check box is greyed out - this implies that another service, such as Transactions, requires this facility.

You can also see the *Must be activated in caller's context* that we mentioned earlier.

A final note on the last checkbox, *Mark component private to application*. Previous versions of COM+ made all configured components publicly available. However, COM+ developers wanted the ability to control access to a component so that it could only be used within a single application (for example, it is a supporting object).

Checking this box will prevent any client outside of this application from being able to instantiate an object of this type.



COM+ Authorisation

QA-IQ

User Groups

Windows NT Access Token

Roles

Policy

Client

Principal

Method Call

Request

COM+ Executive

Reference Monitor

Resource

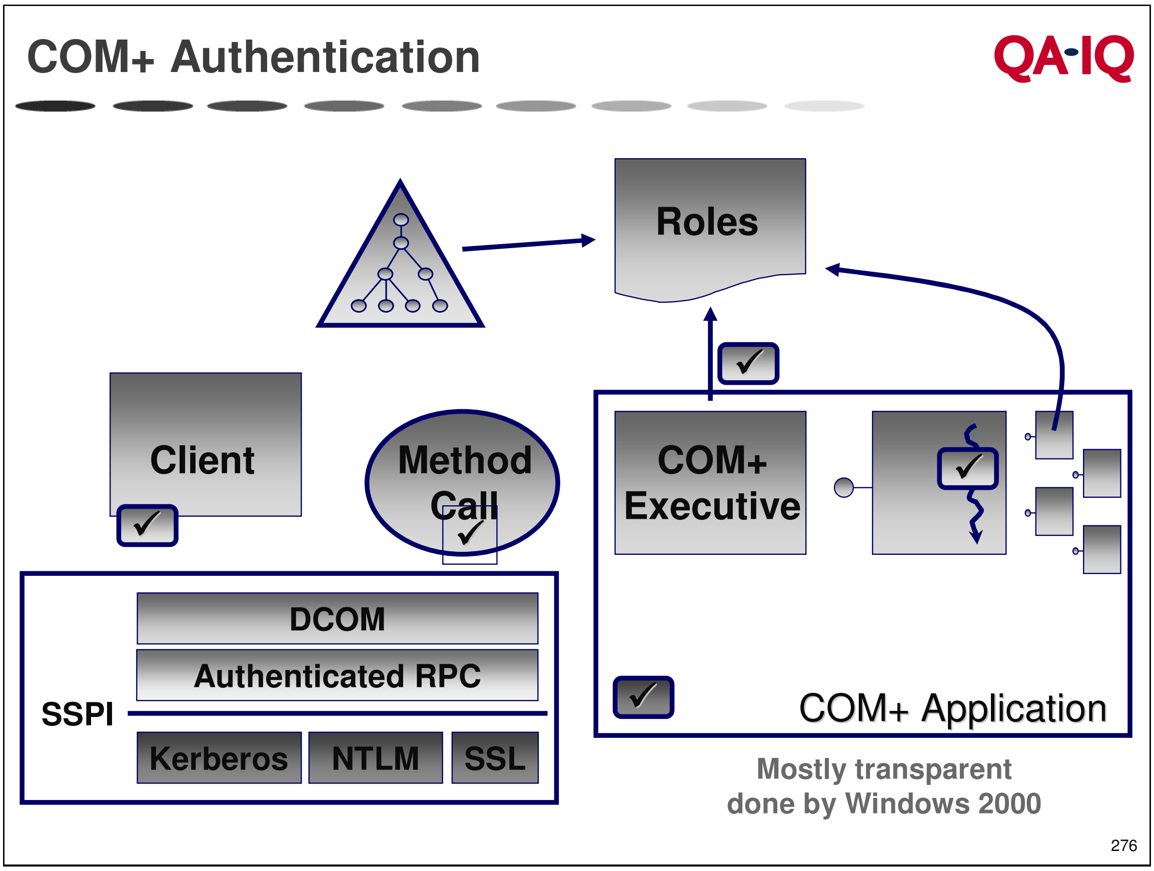
COM+ Application

275

COM+ provides role-based security to control access to COM+ configured components down to the method level.

Every time a call is made, COM+ can validate that caller's security principals and either accept or reject the call.

The concept of the role is that you can cluster Windows groups and users into a single role, such as Managers, and then you can grant access privileges to the roles rather than the individual Windows groups and users.



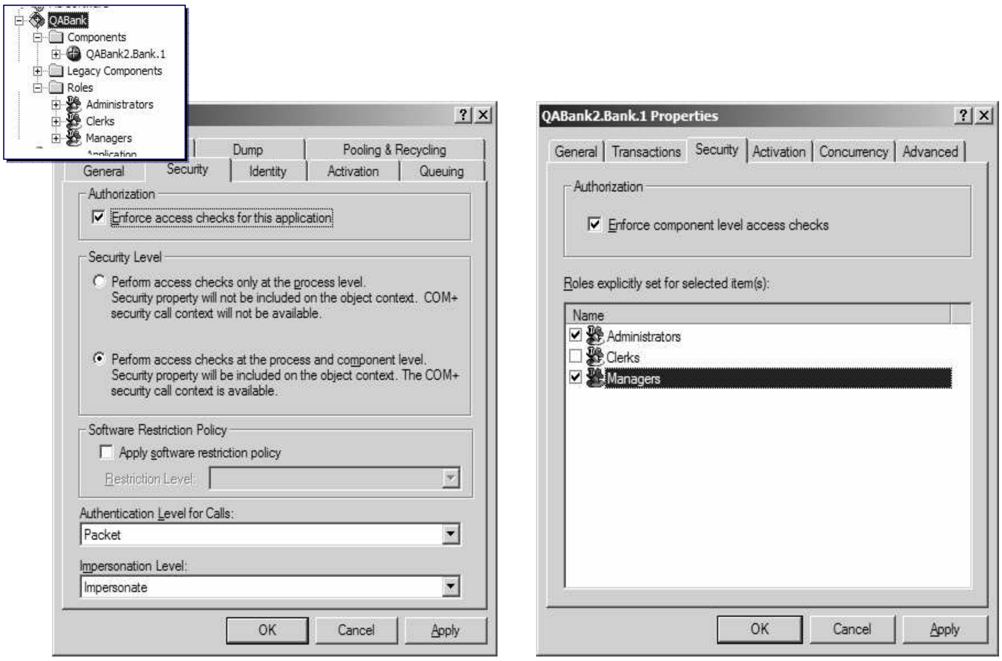
COM+ Authentication

Mostly transparent done by Windows 2000

Windows 2000 also introduced support for a security service provider called Kerberos. This is more advanced than the old NTLM authentication mechanism used in previous versions of Windows. It should be noted that NTLM will continue to be supported, as it is required in situations where you are dealing with a mixed Windows NT/ Windows 2000* environment.

* For Windows 2000 read Windows 2000, XP and .NET Server and beyond.

Configuring Security



Authorization

☑ Enforce access checks for this application

Security Level

○ Perform access checks only at the process level.
Security property will not be included on the object context. COM+ security call context will not be available.

● Perform access checks at the process and component level.
Security property will be included on the object context. The COM+ security call context is available.

Software Restriction Policy

□ Apply software restriction policy

Restriction Level:

Authentication Level for Calls: Packet

Impersonation Level: Impersonate

Authorization

☑ Enforce component level access checks

Roles explicitly set for selected item(s):

Name
☑ Administrators
□ Clerks
☑ Managers

277

Configuring security involves working with both the application's and the components' properties.

Firstly, you will need to set the level of security for the application. If all the components within an application will have the same security requirements, then you can control access at the process level. In this case, individual components will not be able to have their own security checks.

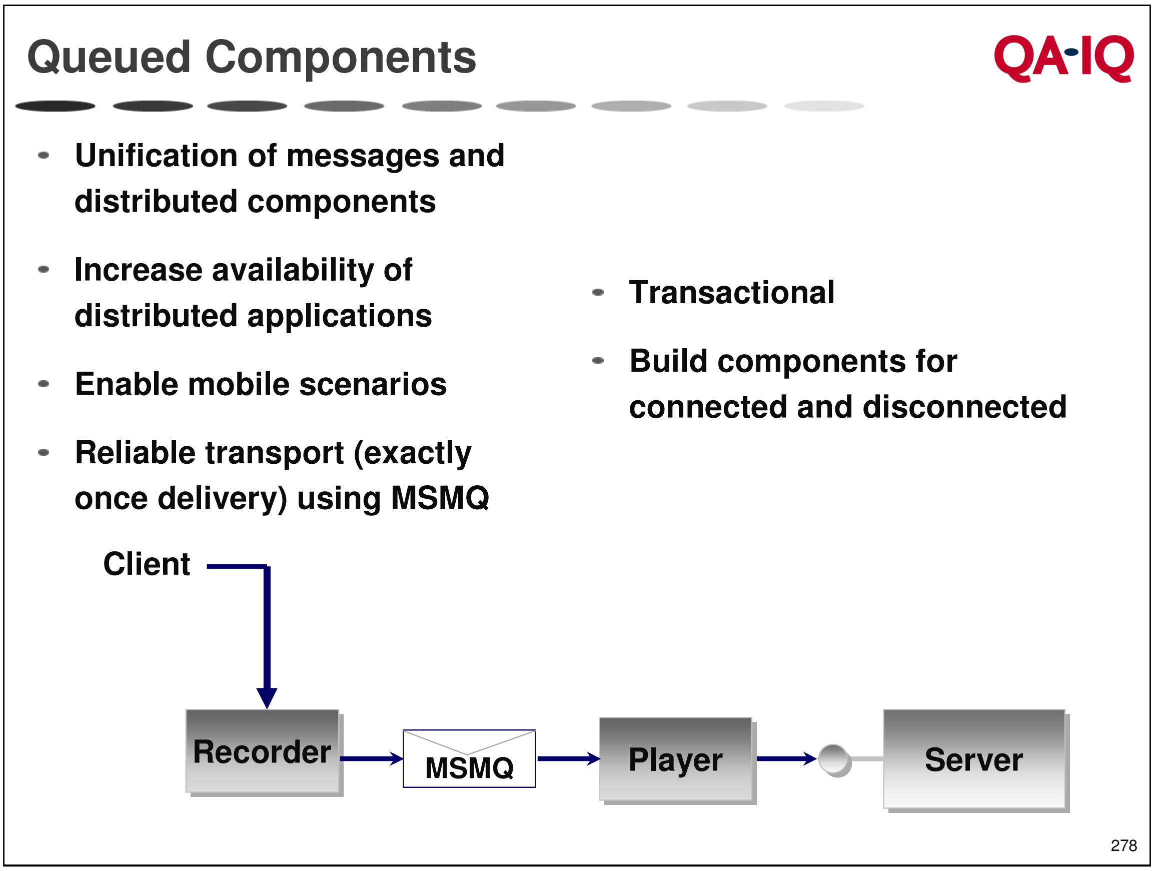
Alternatively, you can configure access at the component and process level. In this case, you can then enforce access checks for each component as needed, and assign access rights to different roles.

Note: If you look at the *Security* tab of a component, and find that everything is greyed out then you should check the following:

(1) Have you enabled security checking in the application to component level?

(2) Are you running the application under a proper set of security credentials*, or are you using the interactive user (this is set on the application's *Identity* tab).

This behaviour is very similar to that as described in the chapter on *Distributed COM Fundamentals*.



Queued Components

QA-IQ

- Unification of messages and distributed components
- Increase availability of distributed applications
- Enable mobile scenarios
- Reliable transport (exactly once delivery) using MSMQ
- Transactional
- Build components for connected and disconnected

278

Queued components are the COM+ way of talking to MSMQ.

MSMQ is a robust, transacted message queuing system, that guarantees that a message will be delivered once (and only once), even if the machine should suffer a traumatic shutdown. However, it does not guarantee **when** the message will arrive.

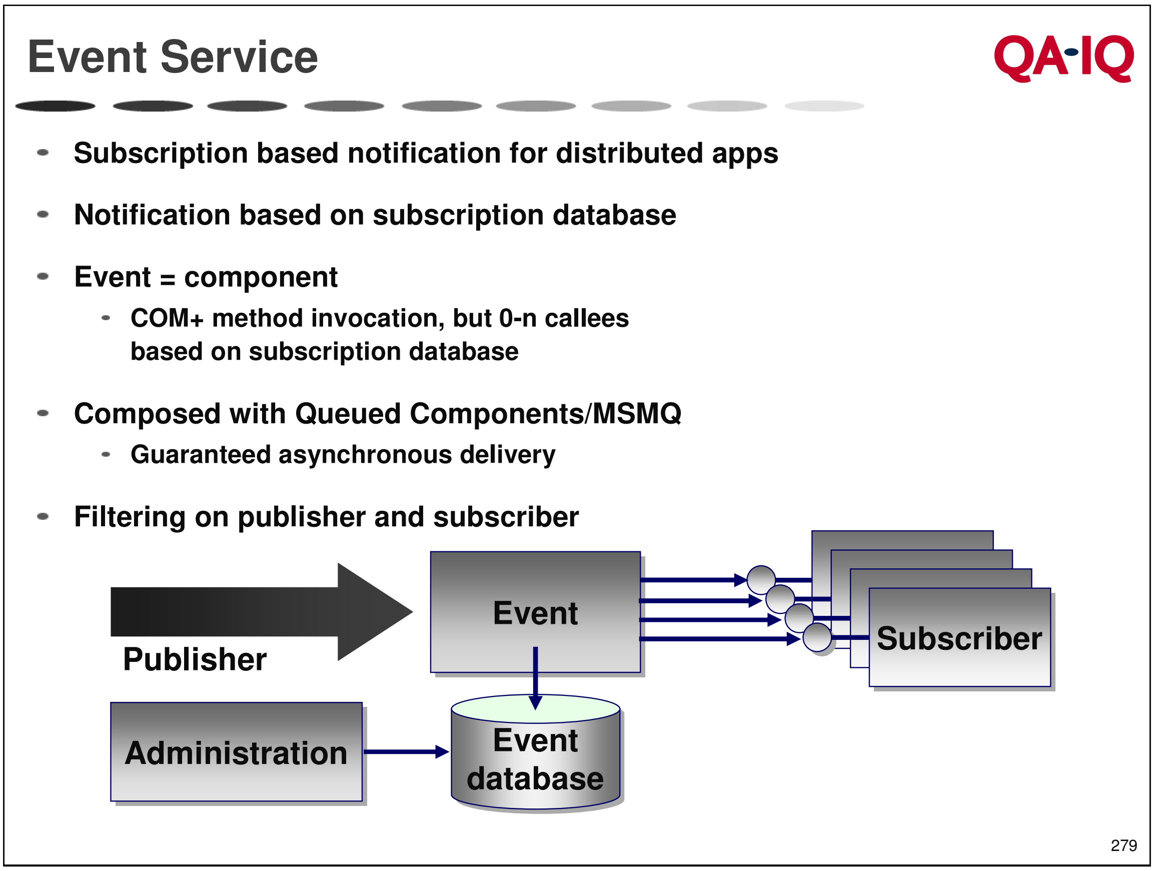
Queued components can be used in a number of scenarios. Consider the case of a travelling salesperson who takes an order from a customer on her laptop. She fills in all the information on her client machine, presses the button and gets the order acknowledgement.

On return to the corporate domain, she fires up the application and the message is delivered silently to the corporate application. The order is processed in the main application, and an acknowledgement is returned to her laptop application.

An alternative scenario might involve a web application where customers can order over the web. Instead of directly connecting the application to the corporate database, queued components are used to communicate between the web server and the application server.

Even if the application server fails, orders can still be taken on the web server as they will be written into a queue. This means that the customers of the site are not affected. As soon as the application server is back online, it can quickly process the orders from the queue.

Note that this scenario wouldn't work if you were only prepared to accept on order on the basis that the requested item was in stock - which would require continual access to the inventory database.



Event Service

QA-IQ

- Subscription based notification for distributed apps
- Notification based on subscription database
- Event = component
  - COM+ method invocation, but 0-n callees based on subscription database
- Composed with Queued Components/MSMQ
  - Guaranteed asynchronous delivery
- Filtering on publisher and subscriber

Publisher

Event

Subscriber

Administration

Event database

279

There are certain situations where you would like to record things that happen to your components, or where you want to notify other components that something has occurred. For this, COM+ introduces the concept of *loosely coupled events*.

This follows a publisher / subscriber model, with a publisher registering an event class with COM+. This is nothing more than a definition of the methods that will be called by the publisher. Subscribers can then find out about the methods, and provide listeners.

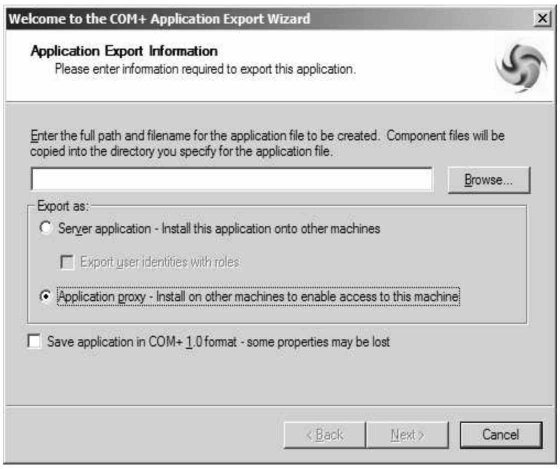
When the publisher fires the event, the subscribers will receive them. In COM+, events can be delivered using queued components, which makes them effectively asynchronous, but with guaranteed delivery.

Examples of when you might want to use loosely coupled events might be in an audit trailing system (who wrote what into the corporate database) or as a management alerting mechanism (disk capacity is now less than 10% of the total available).

Deploying COM+ Applications

QA-IQ

- You can export you application
  - Either the whole thing
  - Or an application proxy
- Produces an .msi Windows Installer File



Welcome to the COM+ Application Export Wizard

Application Export Information

Please enter information required to export this application.

Enter the full path and filename for the application file to be created. Component files will be copied into the directory you specify for the application file.

Export as:

○ Server application - Install this application onto other machines

□ Export user identities with roles

● Application proxy - Install on other machines to enable access to this machine

□ Save application in COM+ 1.0 format - some properties may be lost

280

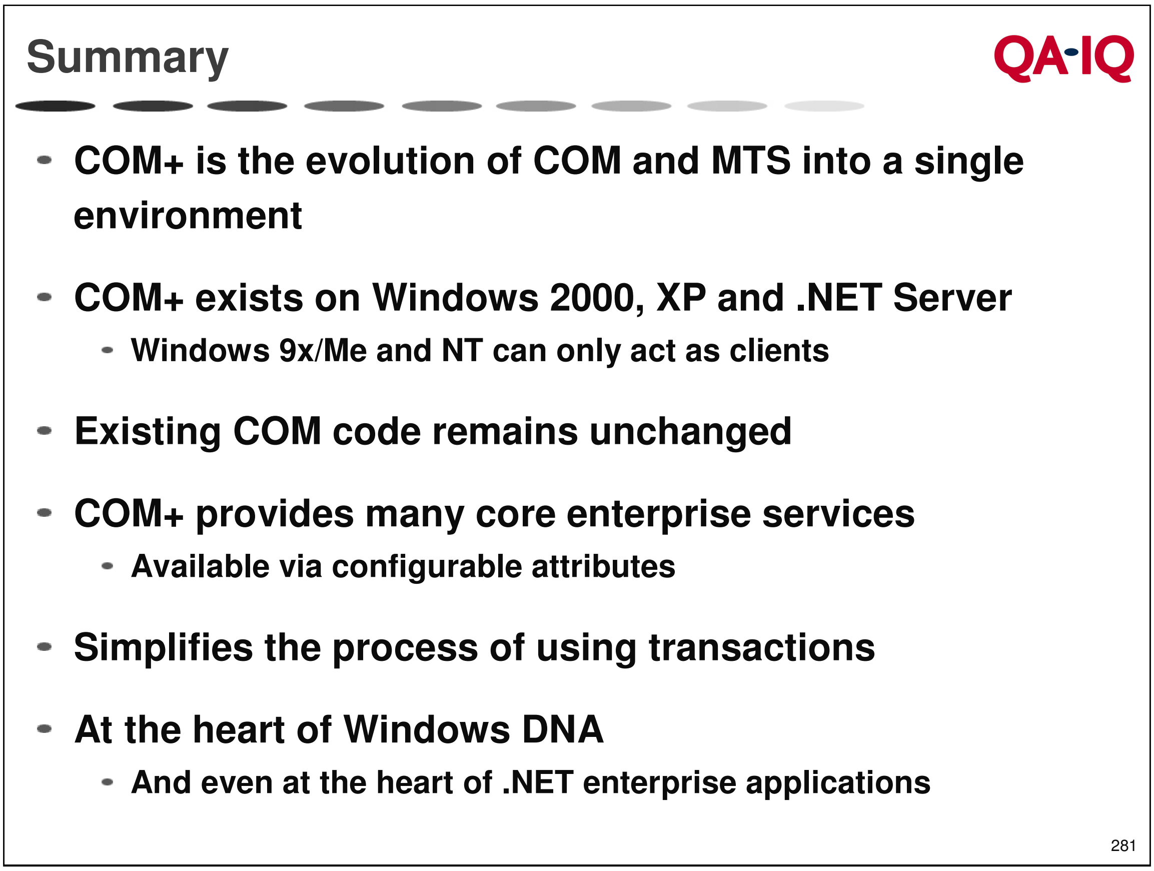
So you've got your COM+ application written and tested. Now you want to deploy it.

Do you really have to go through the whole configuration process on another server? What about just hooking up some clients to it? Fortunately, deployment is relatively straightforward. Right click on the application and choose *Export...*, and away you go.

You can produce two different deployment types, although both types use Microsoft's Windows Installer technology.

The first option is the full Server application. This is the option you select to move an application from server to server. For example, you might develop on one server, and then deploy onto a test server, before finally moving into production.

The second option is the application proxy. This lets you drop the information down onto a second machine, so that when it makes requests they are forwarded to the server. You might select this option when hooking together a web server to an application server.



Summary

QA-IQ

- COM+ is the evolution of COM and MTS into a single environment
- COM+ exists on Windows 2000, XP and .NET Server
  - Windows 9x/Me and NT can only act as clients
- Existing COM code remains unchanged
- COM+ provides many core enterprise services
  - Available via configurable attributes
- Simplifies the process of using transactions
- At the heart of Windows DNA
  - And even at the heart of .NET enterprise applications

281

In this chapter we have taken a quick look at COM+, and the services that it provides for us.

COM+ is the heart of enterprise application architecture under Windows 2000 and beyond.

COM+ is also the evolution (merging) of COM and MTS. Whilst some of this chapter's contents applies to MTS, writing and using configured components under MTS is very different from that in COM+.