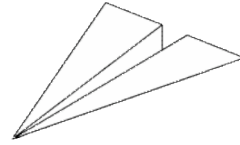


Fundamentals Of COM(+) (Part 2)



Don Box
Cofounder
DevelopMentor
<http://www.develop.com/dbox>

11-204

DEVELOPMENTOR

COM – The idea

- ❖ **COM is based on three fundamental ideas**
- ❖ **Clients program in terms of interfaces, not classes**
- ❖ **Implementation code is not statically linked, but rather loaded on-demand at runtime**
- ❖ **Object implementers declare their runtime requirements and the system ensures that these requirements are met**
- ❖ **The former two are the core of classic COM**
- ❖ **The latter is the core of MTS and COM+**

Tale Of Two COMs

- ❖ **COM is used primarily for two tasks**
- ❖ **Task 1: Gluing together multiple components inside a process**
 - **Class loading, type information, etc**
- ❖ **Task 2: Inter-process/Inter-host communications**
 - **Object-based Remote Procedure Calls (ORPC)**
- ❖ **Pros: Same programming model and APIs used for both tasks**
- ❖ **Cons: Same programming model and APIs used for both tasks**
- ❖ **Design around the task at hand**

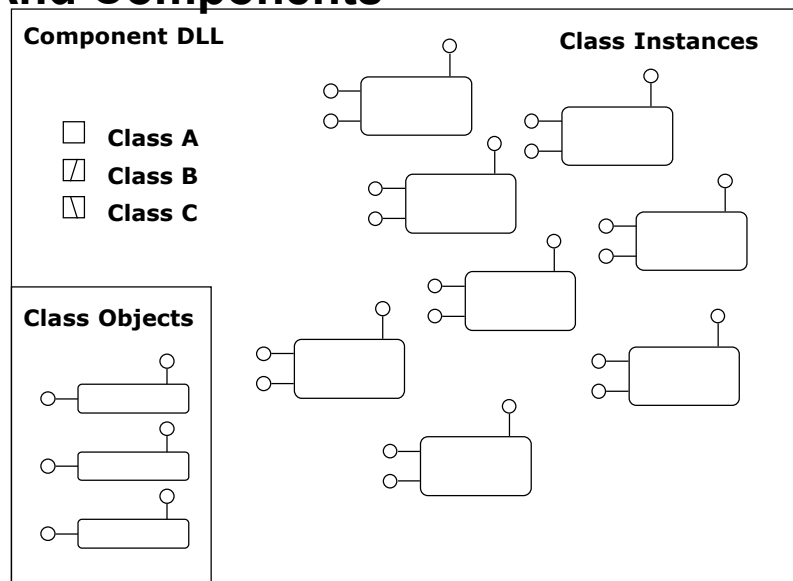
Definitions

- ❖ **Two key terms have been defined so far**
- ❖ **A COM Interface is a collection of abstract operations one can perform on an object**
 - **Must extend IUnknown directly or indirectly**
 - **Identified by a UUID (IID)**
 - **Platform-specific vptr/vtable layout**
- ❖ **A COM Object is a collection of vptrs in memory that follow the COM identity laws**
 - **Must implement at least one COM interface**
 - **QueryInterface ties vptrs together into cohesive object**
- ❖ **Objects assumed to materialize from thin air!**

Definitions

- ❖ A COM Class (or coclass) is a named body of code that can be used to produce COM objects
- ❖ All coclasses are named by a UUID (CLSID)
- ❖ All coclasses have a distinguished object that is used to create new instances
 - Called a class object or class factory
 - Typically implements IClassFactory
- ❖ All coclasses loaded on demand by class loader
 - Called the Service Control Manager or (SCM)
- ❖ For efficiency, a single component DLL can support multiple COM classes

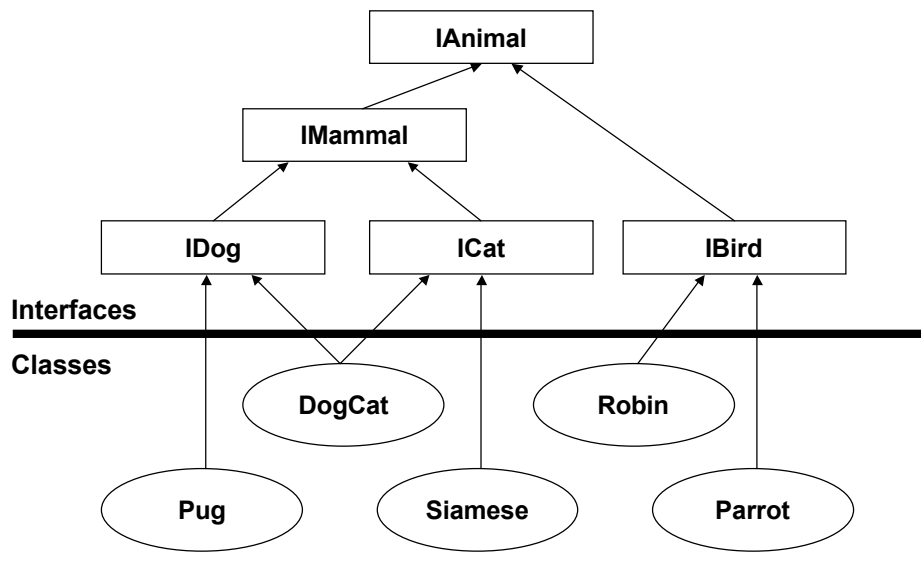
Classes, Class Objects, And Components



Class Versus Type

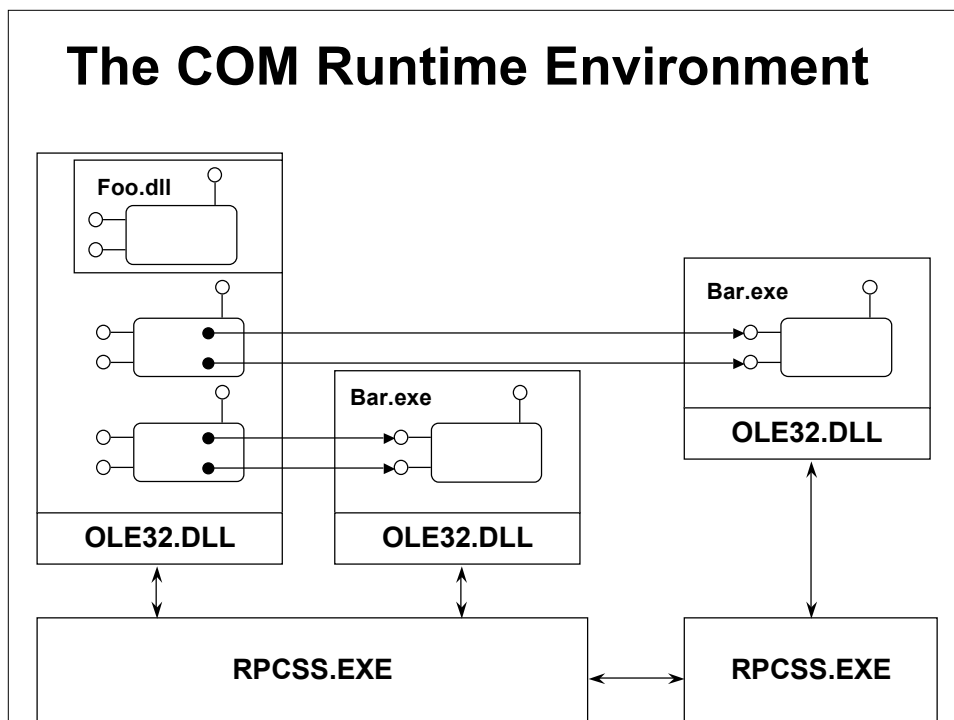
- ❖ **An Interface represents a data type suitable for declaring variables**
 - Non-trivial operations
 - Hierarchical with respect to one another
 - Polymorphic with respect to different objects
- ❖ **A Class represents loadable concrete code used to create objects**
 - Resultant objects implement one or more interfaces
- ❖ **Class unsuitable for declaring variables**
 - Entire motivation for interface-based programming based on relative uselessness of class

Class Versus Type



The COM Runtime Environment

- ❖ The infrastructure used to support COM on a given platform is called the COM library
- ❖ Each thread that uses COM library must setup/teardown thread-specific data structures
 - CoInitialize[Ex] and CoUninitialize do this for you
- ❖ The COM library implemented in several DLLs
 - OLE32.DLL – core class/interface functionality
 - OLEAUT32.DLL – Visual Basic®-centric type infrastructure
- ❖ Inproc class loading done in OLE32.DLL
- ❖ Cross-process/host class loading performed by Windows NT® Service (RPCSS.EXE)



COM Class Loading

- ❖ Clients issue activation calls against the SCM
- ❖ SCM responsible for locating component and loading it into memory
- ❖ SCM queries component for class object and (optionally) uses it to instantiate new instance
- ❖ Once SCM returns a reference to class instance/class object, SCM out of the picture
- ❖ Based on configuration, COM may need to load component in separate process (potentially on different machine)

COM Class Loading And Locality

- ❖ All activation calls allow client to indicate locality
 - SCM chooses most efficient allowed by client

```
CLSCTX_INPROC_SERVER // load in client process
CLSCTX_INPROC_HANDLER // use OLE Document rendering handler
CLSCTX_LOCAL_SERVER // load in separate process
CLSCTX_REMOTE_SERVER // load on distinct host machine
CLSCTX_SERVER // CLSCTX_*_SERVER
CLSCTX_ALL // CLSCTX_*
```

```
typedef struct _COSERVERINFO {
    DWORD dwReserved1; // m.b.z.
    const OLECHAR *pwszName; // host name
    COAUTHINFO *pAuthInfo; // security goo
    DWORD dwReserved2; // m.b.z.
} COSERVERINFO;
```

Using The SCM

- ❖ The SCM exposes two core activation APIs
- ❖ Both APIs load component automatically
- ❖ Both APIs accept a CLSID and information about component location as input parameters
- ❖ CoGetObject returns class object/factory
 - No new instances created
- ❖ CoCreateInstanceEx uses IClassFactory interface on class object to create new instance
 - Class object never returned to client

CoGetObject/IClassFactory

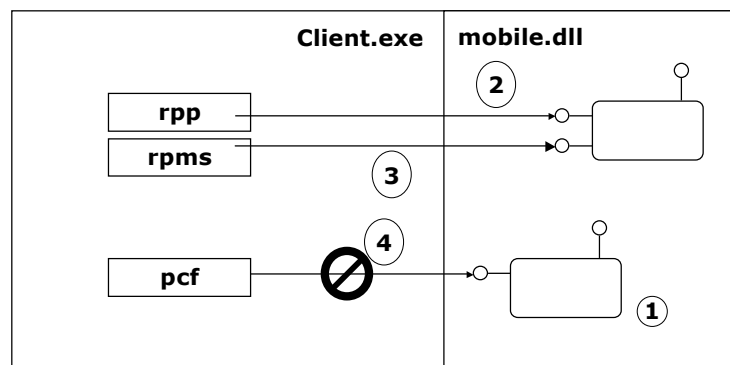
```
interface IClassFactory : IUnknown {  
    // create a new com object  
    HRESULT CreateInstance([in] IUnknown *pUnkOuter,  
                           [in] REFIID riid,  
                           [out,retval,iid_is(riid)] void **ppv);  
    // hold component code in memory  
    HRESULT LockServer([in] BOOL bLock);  
}
```

```
HRESULT CoGetObject(  
    [in] const CLSID& rclsid, // which class?  
    [in] DWORD dwClsCtx,     // locality?  
    [in] COSERVERINFO *pcsi, // host/sec info?  
    [in] REFIID riid,       // which interface?  
    [out, iid_is(riid)] void **ppv // put it here!  
);
```

Example

```
void CreatePager(IPager *&rpp, IMessageSource *&rpms) {
    IClassFactory *pcf = 0; rpp = 0; rpms = 0;
    // ask SCM to load class code for Pager
    HRESULT hr = CoGetClassObject(CLSID_Pager, CLSCTX_ALL, 0,
        IID_IClassFactory, (void**)&pcf);
    if (SUCCEEDED(hr)) {
        // ask class code to create new class instance
        hr = pcf->CreateInstance(0, IID_IPager, (void**)&rpp);
        if (SUCCEEDED(hr))
            hr = rpp->QueryInterface(IID IMessageSource,
                (void**)&rpms);
        pcf->Release();
    }
}
```

Example



- 1 Client calls CoGetClassObject
- 2 Client calls CreateInstance on Class Object
- 3 Client calls QueryInterface on Class Instance
- 4 Client calls Release on Class Object

CoGetObject Pitfalls

- ❖ Previous example made at least four round-trips in distributed case
 - One for CoGetObject
 - One for CreateInstance
 - One for QueryInterface
 - One for IClassFactory::Release
- ❖ Superior solution would perform class loading and object creation in one round trip
- ❖ Solution: CoCreateInstance[Ex]

CoCreateInstanceEx

```
HRESULT CoCreateInstanceEx(  
    [in] const CLSID& rclsid, // which class?  
    [in] IUnknown *pUnkOuter, // used in aggregation  
    [in] DWORD dwClsCtx, // locality  
    [in] COSERVERINFO *pcsi, // (opt) host/sec. info  
    [in] ULONG cltfs, // # of interfaces  
    [in, out] MULTI_QI *prgmqi // put them here!  
);
```

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

```
HRESULT CoCreateInstance(  
    [in] const CLSID& rclsid, // which class?  
    [in] IUnknown *pUnkOuter, // used in aggregation  
    [in] DWORD dwClsCtx, // locality?  
    [in] REFIID riid, // which interface?  
    [out, iid_is(riid)] void **ppv // put it here!  
);
```

Example

```
void CreatePager(IPager *&rpp, IMessageSource *&rpms) {
    rpp = 0; rpms = 0;
    // build vector of interface requests
    MULTI_QI rgmqi[] = { { &IID_IPager, 0, 0 },
                        { &IID IMessageSource, 0, 0 } };
    // ask COM to load class code and create instance
    HRESULT hr = CoCreateInstanceEx(CLSID_Pager,
                                    0, CLSCTX_ALL, 0, 2, rgmqi);
    // extract interface pointers from rgmqi vector
    if (SUCCEEDED(hr)) {
        if (hr == S_OK || SUCCEEDED(rgmqi[0].hr))
            rpp = reinterpret_cast<IPager*>(rgmqi[0].pIif);
        if (hr == S_OK || SUCCEEDED(rgmqi[1].hr))
            rpms = reinterpret_cast<IMessageSource*>(rgmqi[1].pIif);
    }
}
```

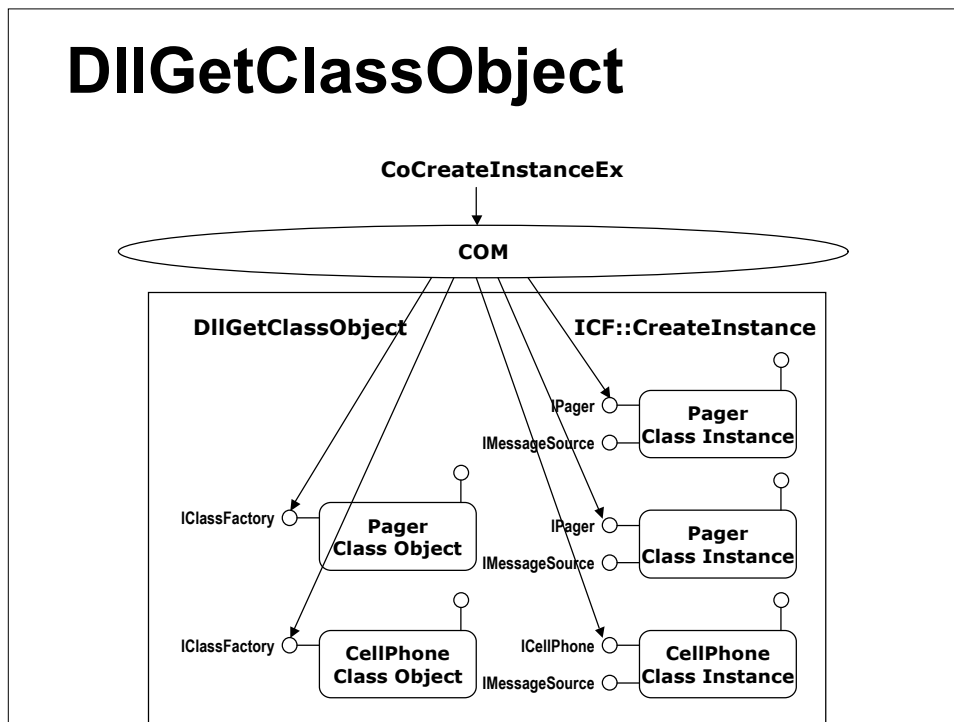
Exposing COM Classes

- ❖ **Component DLLs export a well-known function used by COM to extract class object from DLL**

```
STDAPI DllGetClassObject(
    [in] REFCLSID rclsid, // which class?
    [in] REFIID riid, // which interface?
    [out, iid_is(riid)] void **ppv // put it here!
);
```

- ❖ **DllGetClassObject called by CoGetClassObject and CoCreateInstance[Ex] to access class object**
 - Never called directly by client code
- ❖ **If DLL doesn't export DllGetClassObject, all activation calls will fail**

DllGetClassObject



Exposing Class Objects/Inproc

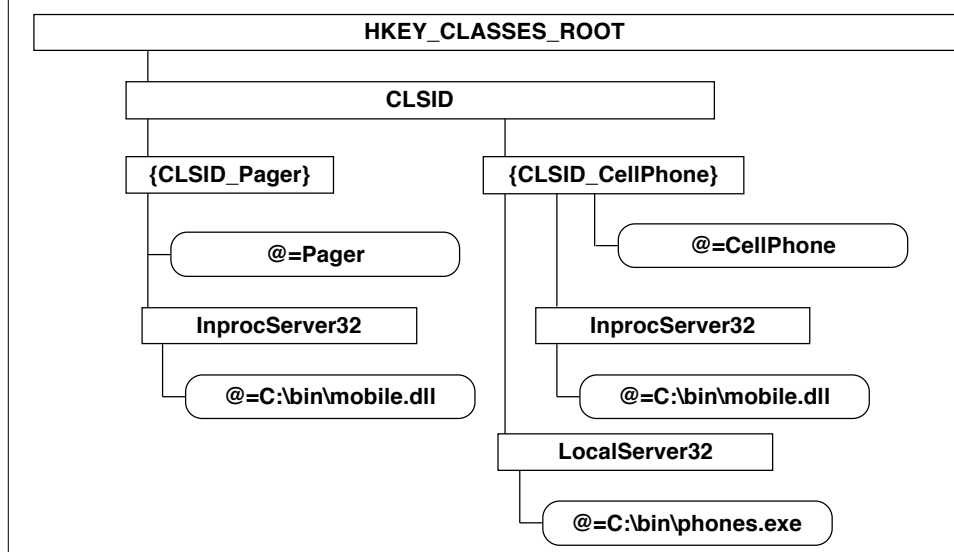
```
PagerClassObject g_coPager;  
CellPhoneClassObject g_coCellPhone;  
  
STDAPI DllGetClassObject(REFCLSID rclsid,  
                          REFIID riid, void**ppv) {  
    if (rclsid == CLSID_Pager)  
        return g_coPager.QueryInterface(riid, ppv);  
    else if (rclsid == CLSID_CellPhone)  
        return g_coCellPhone.QueryInterface(riid, ppv);  
    *ppv = 0;  
    return CLASS_E_CLASSNOTAVAILABLE;  
}
```



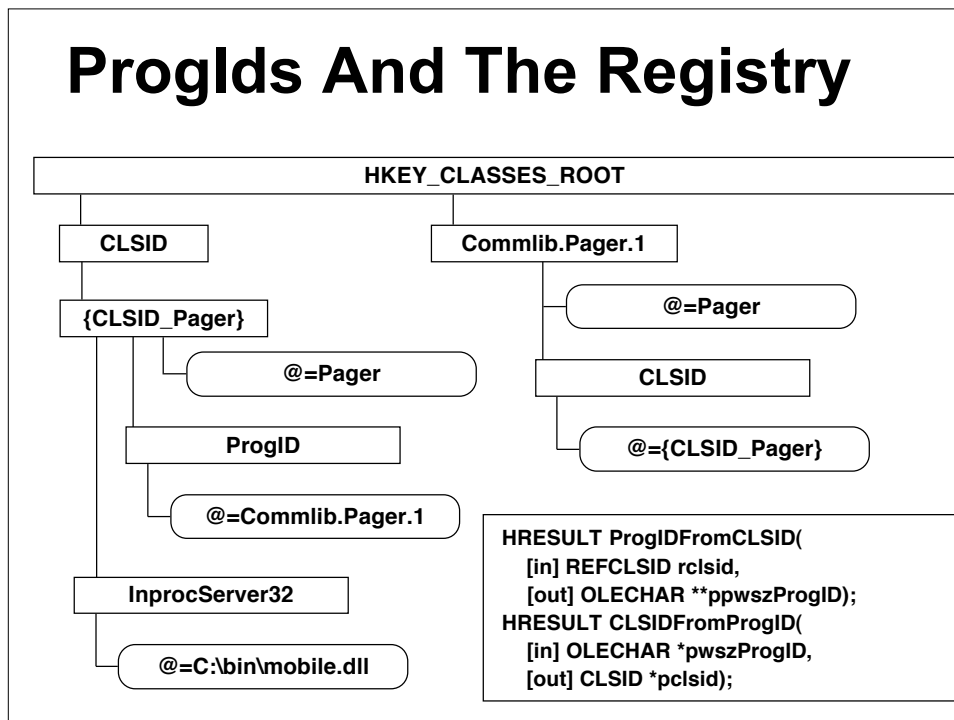
Finding Components

- ❖ All COM classes registered under distinguished key in registry (HKEY_CLASSES_ROOT)
 - Holds machine-wide configuration under Windows NT 4.0
 - Magic key under W2K that merges machine-wide registration with current user's private configuration
- ❖ Can also register text-based aliases for CLSIDs called ProgIDs for GUID-hostile environments
- ❖ REGSVR32.EXE used to install component DLLs that export two well-known entry points
 - STDAPI DllRegisterServer(void);
 - STDAPI DllUnregisterServer(void);

CLSID And The Registry



ProgIDs And The Registry



COM Classes And IDL

- ❖ COM classes can be declared in IDL using coclass statement
- ❖ Coclass statement generates class entry in TLB
- ❖ Coclass statement generates CLSID_XXX variables in generated C(++) headers
 - Generates __declspec(uuid) statements as well
- ❖ Coclass statement allows minimum supported interfaces to be listed as well

Async Calls

```

[ uuid(03C20B33-C942-11d1-926D-006008026FEA) ]
coclass Pager {
    [default] interface IPager;
    interface IMessageSource;
}
    
```

Interception

- ❖ **In general, it is better to leverage platform code than to write it yourself**
 - Thread scheduler, file system, window manager
- ❖ **Classically, the platform has been exposed through explicit APIs and interfaces**
 - Requires some code on your part to utilize
- ❖ **COM is moving towards exposing the platform through interception**
 - COM puts a middleman between the client and object
 - Middleman makes calls to the platform on object's behalf both before and after object's method executes

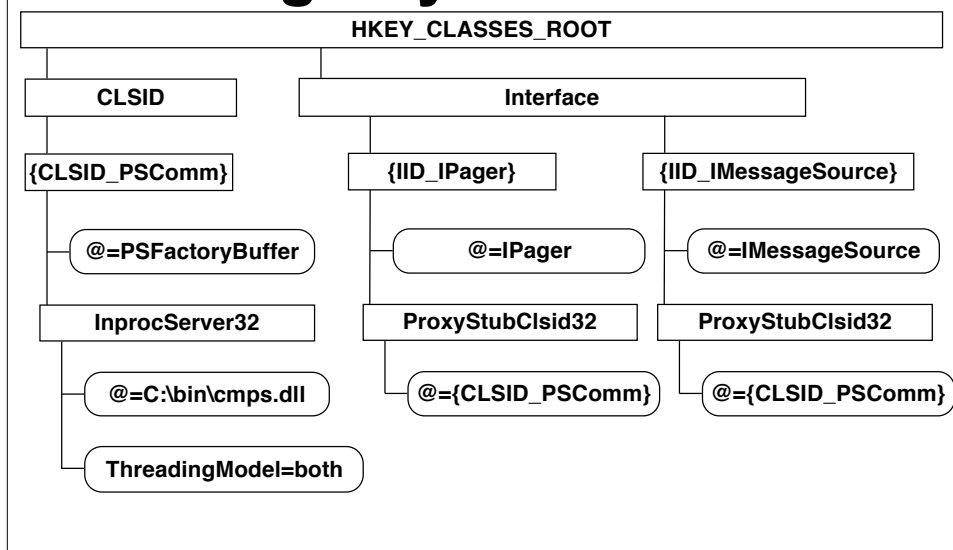
Interception Basics

- ❖ **To provide a service, the system must intercept all calls to your object**
- ❖ **Interceptors pre- and post-process every call**
 - Interceptors make system calls on your behalf
 - Interceptors set up the runtime environment for your method calls
 - Interceptors may fail the method call without your participation
- ❖ **Interceptors must know what your interfaces look like**
 - All interfaces exposed by configured components require specially prepared P/S DLL or a type library

Interception And Interfaces

- ❖ Interception needs type info for all interfaces
- ❖ Interfaces marked [dual] and [oleautomation] can simply rely on type library
 - Parameter types limited to VARIANT-compatible
- ❖ Interfaces not marked [dual] or [oleautomation] require a specially prepared proxy/stub DLLs
 - Run MIDL compiler using /Oicf flag
 - Compile foo_i.c, foo_p.c and dllhost.c using /MD switch
 - Link P/S dll against MTXIH.LIB, OLE32.LIB and ADVAPI32.LIB before any other libraries
- ❖ Registering P/S DLL or (dual/oleautomation) TLB inserts entries under HKCR\Interfaces

Proxy/Stub DLLs And The Registry

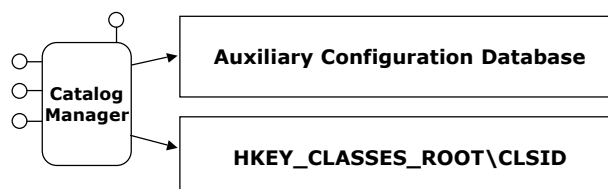


Configured Components

- ❖ **Problem:** If goal is to write little or no code, how do we configure interceptor to do its magic?
- ❖ **Solution:** Declarative attributes
- ❖ **Classes that require extended services must indicate this declaratively**
- ❖ **COM+/MTS introduce the notion of configured components**
- ❖ **Configured components are classes that have extended attributes that control interception**
- ❖ **Configured components always DLLs**
 - MTS/COM+ use surrogate for remote/local activation

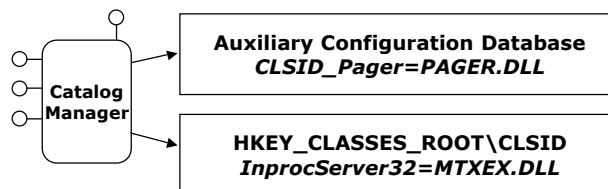
Configured Components

- ❖ **MTS and COM+ have radically different details wrt how configuration information is stored and used**
- ❖ **Both use HKEY_CLASSES_ROOT\CLSID**
- ❖ **Both store information in auxiliary storage**
- ❖ **Details abstracted away behind catalog manager**



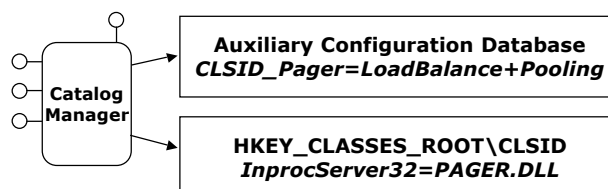
Configured Components - MTS Style

- ❖ MTS layers on top of classic COM
- ❖ Runtime services provided by MTS executive
 - Lives in MTXEX.DLL
- ❖ MTS CatMan stores MTXEX.DLL under HKCR to ensure MTS gets between client and object
 - Stores component filename in aux database



Configured Components - COM+ Style

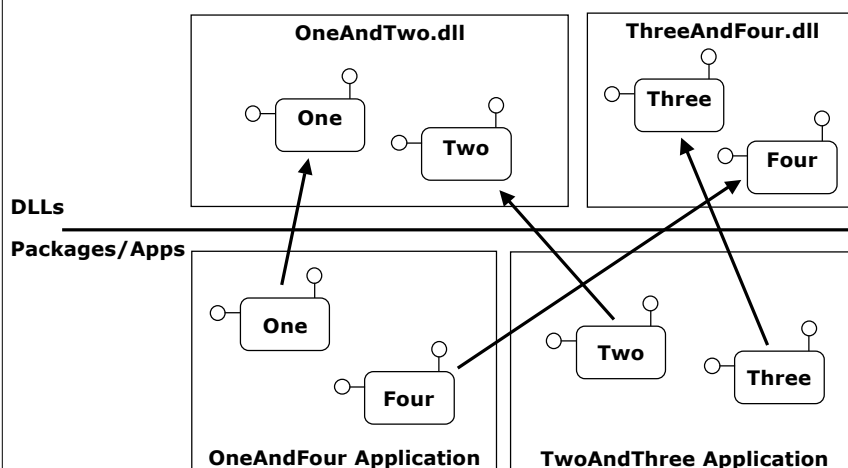
- ❖ Under COM+, runtime services provided by COM itself
- ❖ CoCreateInstance is smart enough to consult auxiliary information at activation-time
- ❖ COM+ CatMan stores still manages extended attributes in auxiliary database



Packages/Applications

- ❖ The catalog manager segregates classes into COM+ applications (or MTS packages)
- ❖ Each configured class belongs to exactly one application
- ❖ All classes in an application share activation settings
- ❖ Configuration orthogonal to physical packaging
 - x classes from y DLLs mapped into z applications
- ❖ Applications can be configured to load in activator's process (library) or in distinct surrogate process (server)

x Classes, y DLLs, z Applications/Packages



Attributes

- ❖ The catalog stores attributes that the runtime interrogates to build an interceptor
- ❖ The set of attributes is fixed (for now)
- ❖ Applications/packages, classes, interfaces and methods can all have attributes
- ❖ Can set attributes using COM+/MTS explorer
- ❖ Will be able to set all attributes from development environment someday...

Attributes: Applications/Packages

Activation Type	<u>Library (inproc)/Server (surrogate)</u>
Authentication Level	<u>None, Connect, Call, Packet, Integrity, Privacy</u>
Impersonation Level	<u>Identify, Impersonate, Delegate</u>
Authorization Checks	<u>Application Only/Application + Component</u>
Security Identity	<u>Interactive User/Hardcoded User ID + PW</u>
Process Shutdown	<u>Never/N minutes after idle</u>
Debugger	Command Line to Launch Debugger/Process
Enable Compensating Resource Managers	On/ <u>Off</u>
Enable 3GB Support	On/ <u>Off</u>
Queueing	Queued/Queued+Listener

Underlines indicate settings available under MTS

Attributes: Classes, Interfaces, And Methods

Transaction	<u>Non Supported</u> , <u>Supported</u> , <u>Required</u> , <u>Requires New</u>	Class
Synchronization	<u>Non Supported</u> , <u>Supported</u> , <u>Required</u> , <u>Requires New</u>	Class
Object Pooling	<u>On/Off</u> , Max Instances, Min Instances, Timeout	Class
Declarative Construction	Arbitrary Class-specific String	Class
JIT Activation	<u>On/Off</u>	Class
Activation-time Load Balancing	<u>On/Off</u>	Class
Instrumentation Events	<u>On/Off</u>	Class
Declarative Authorization	<u>Zero or more role names</u>	Class Interface Method
Auto-Deactivate	<u>On/Off</u>	Method
Must Activate in Activator's Context	<u>On/Off</u>	Class

Underlines indicate settings available under MTS

Exporting Packages/ Applications

- ❖ MTS/COM+ allow package/app configuration to be exported to the file system for distribution
- ❖ MTS: Exporting produces a .PAK file that contains snapshot of catalog for the package
 - Also contains flattened references to all DLLs/TLBs
- ❖ COM+: Exporting produces a single .MSI file that contains both catalog info and DLL/TLBs
- ❖ .PAK/.MSI file can be imported on other host machines
 - Can be done remotely using remote catalog access

Package/Application Export Residue

MTS

MYAPP.PAK
Catalog info

MYCOMP1.DLL
code for some classes

MYCOMP2.DLL
code for other classes

MYPS.DLL
proxy/stub code

COM+

MYAPP.MSI
Catalog info

MYCOMP1.DLL
code for some classes

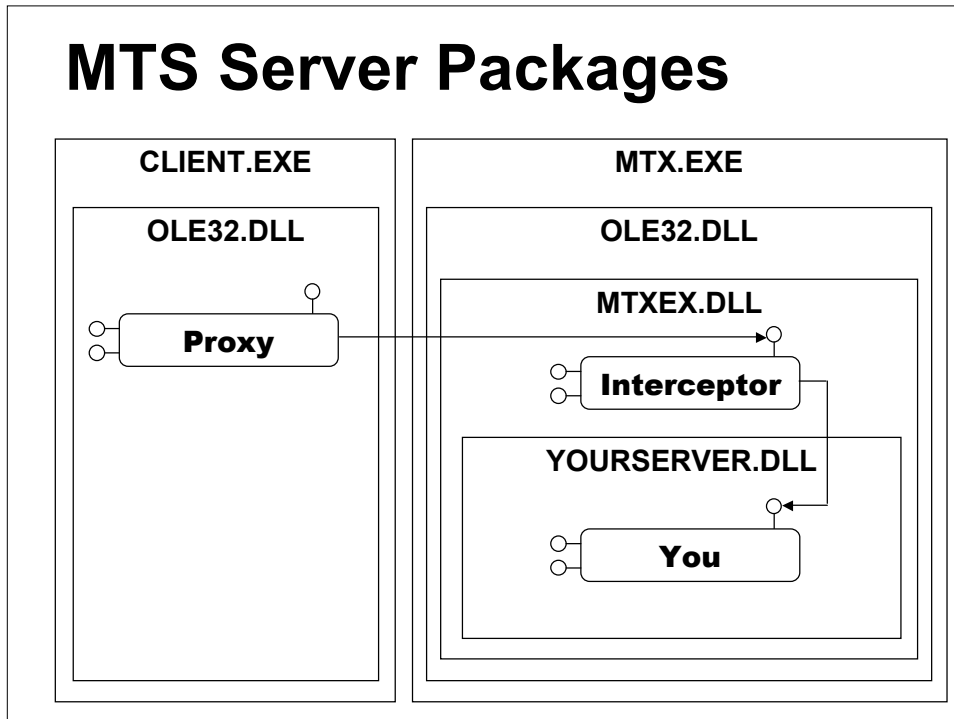
MYCOMP2.DLL
code for other classes

MYPS.DLL
proxy/stub code

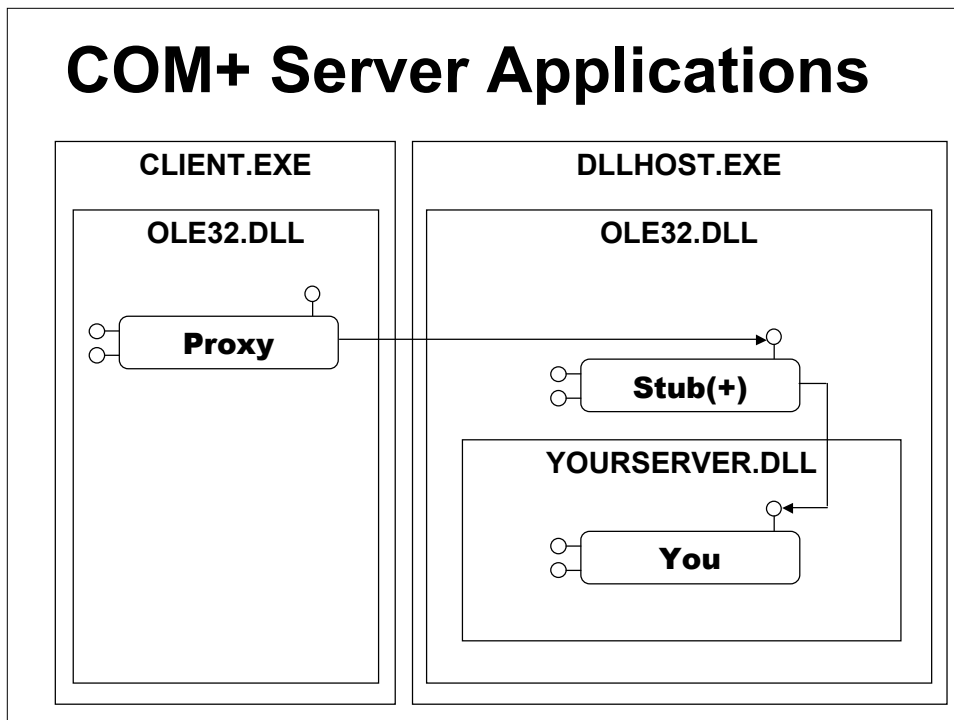
Server Packages/ Applications

- ❖ An application can be configured to activate as a library application or a server application
 - Server applications are the norm in MTS/COM+
- ❖ Only server applications support...
 - Remote activation
 - Complete Security Support
 - Insulating user of component from component faults
- ❖ MTS Server packages are loaded by the MTS Surrogate (mtx.exe)
- ❖ COM+ Server packages are loaded by default COM surrogate (dllhost.exe)
 - dllhst3g.exe if 3GB support is enabled in catalog

MTS Server Packages



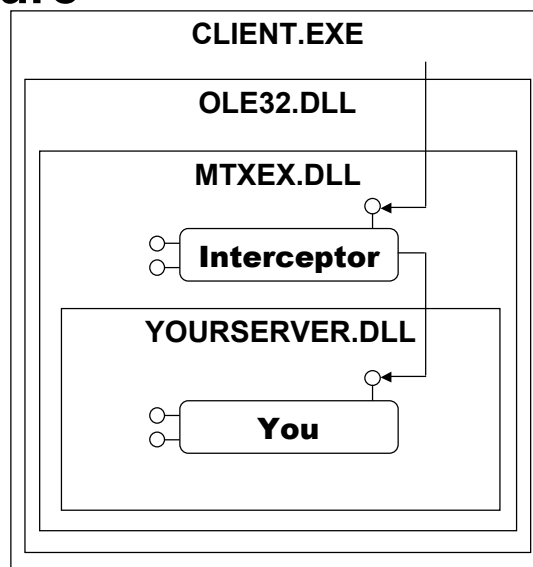
COM+ Server Applications



Library Applications/Packages

- ❖ **Library applications/packages load in the creator's process**
 - Solves the "1 class used by 3 applications" problem
- ❖ **MTS catalog manager controls registry entries for components in library packages**
 - Each class's InprocServer32 key points to the MTS Executive (mtxex.dll)
 - MTS Executive creates interceptor between client and object based on catalog info
 - MTS Executive manages a thread pool to service activation calls and general housekeeping
- ❖ **Instances will always always be protected from concurrent access under MTS!**

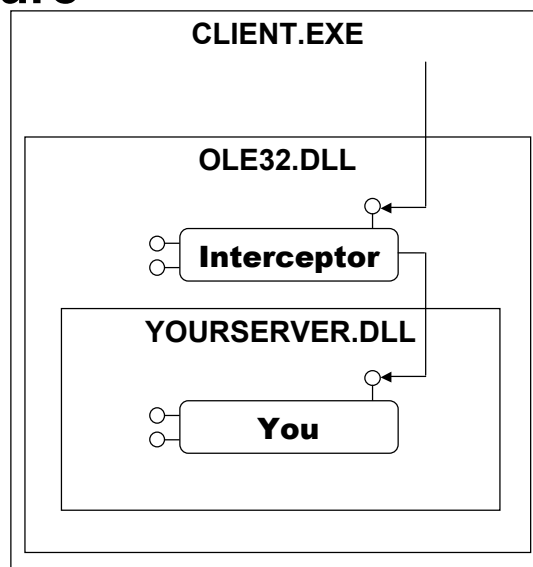
MTS Library Packages In Nature



How COM(+) Library Applications Work

- ❖ **COM+ catalog manager leaves InprocServer32 entry alone**
 - **Additional attributes stored in aux config database**
- ❖ **CoCreateInstance checks for extended attributes and creates an interceptor as needed**
- ❖ **Instances may or may not be protected from concurrent access depending on configuration!**
 - **Default setting at install-time is protected, but can easily defeat using COM+ Explorer**

COM+ Library Applications In Nature



Summary

- ❖ The SCM dynamically loads COM class code
- ❖ COM+ and MTS exposes services through interception
- ❖ Components configure their interceptors through declarative attributes stored in a configuration database
- ❖ MTS/COM+ consult configuration database at activation time
- ❖ Classes are grouped into applications/packages
- ❖ The catalog is a scriptable MTS/COM+ component

References

- ❖ **Programming Dist Apps With Visual Basic and COM**
 - Ted Pattison, Microsoft Press
- ❖ **Inside COM**
 - Dale Rogerson, Microsoft Press
- ❖ **Essential COM(+), 2nd Edition (the book)**
 - Don Box, Addison Wesley Longman (4Q99)
- ❖ **Essential COM(+) Short Course, DevelopMentor**
 - <http://www.develop.com>
- ❖ **DCOM Mailing List**
 - <http://discuss.microsoft.com>

Where do you want to go today?®

Microsoft