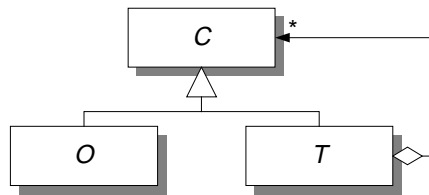


*Evaluation of COM Support in Visual
C++ using ATL
COT/3-7-V1.0*



Centre for Object Technology

Revision history: V1.0 19-08-98 First version.

Author(s): Kristian Lippert, DTI

Status: Final.

Publication: Public

Summary:

This document presents the answer to a number of specific questions regarding the support for COM based development using Microsoft Visual C++ and the ATL library.

© Copyright 1998 Danish Technological Institute

Introduction

This document presents the answer to a number of specific questions regarding the support for COM based development using Microsoft Visual C++ using the ATL class library.

This report is part of an evaluation of four development environments with respect to their support for COM based development. The main document of this evaluation is COT-3-4: Evaluation of COM Support in Development Environments.

1. Component Server

1.1 Which types of components does the tool support creation for?
(1.1)

-
- 1.1.1 Support for visual Controls ?
(1.1.1) *The support for visual Controls are concentrated around full controls, IE controls (Internet Explorer) and property pages. Creating custom controls can be done, but much of the work for visibility has to be done through pure Win32 API calls. Subclassing win32 controls can easily be done. The wizards will generate a skeleton that is pretty easy to extend to the wanted usage.*
-
- 1.1.2 Support for non-visual Controls ?
(1.1.2) *1.1.2 The interesting standard controls that can be generated are custom components, Add-ins, IE objects, ActiveX Server Component, MS Transaction Server component. The custom component are the fundamental building block from which all other components (potentially) can be generated. It supports IUnknown and IDispatch derived interfaces.*
-

1.2 Types of servers
(1.2)

-
- 1.2.1 Can the tool generate/create In-process servers?
(1.2.1) *Yes*
-
- 1.2.2 Can the tool generate/create Out-of-process servers?
(1.2.2) *Yes*
-
- 1.2.3 How can you make transition between in-process and out-of-process servers?
(1.2.3)
 - 1.2.3.1 You can do it through your IDE by clicking in some menu (where) ?
(1.2.3.1) *No*
-

- 1.2.3.2 Changing the code by hand ?
(1.2.3.2) Maybe
-

- 1.2.3.3 Restarting the project and moving the code manually ?
(1.2.3.3) Yes
-

- 1.2.3.4 Other:
(1.2.3.4) No
-

- 1.2.4 Can multiple components be contained in the same server ?
(1.2.4) Yes
-

- 1.2.5 Are there any limit on the number of different components in a server ?
(1.2.5) Not that I know of.
-

<P

1.3 Component construction (tool support) *(1.3)*

- 1.3.1 How are the COM components created ?
(1.3.1)
-

- 1.3.1.1 Not using wizard. Comments:
(1.3.1.1) If the components are to be generated by hand it is a very tedious job. You can compare it to handcoding a C++ program in assembler code.
-

- 1.3.1.2 using wizard. Comments:
(1.3.1.2) You can generated three different kinds of components: In-Proc, Out-Proc and Services. The Inproc components
-

- 1.3.1.2.1 Which code-generation-wizards exist (list names and usages)?

(1.3.1.2.1) You can choose between three different kinds of in-proc components. These are Objects, Controls, and Miscellaneous

- **Objects:**

- *Simple Objects adds a minimal COM object.*
- *Add-in Objects creates a COM object that can be used to extend Developers Studio.*
- *Internet Explorer Object creates an object that contains all the interfaces necessary for it to work with Internet Explorer, but has no user interface.*
- *ActiveX Server Component creates an object that can be used as a part of an ActiveX Server Page (ASP) with Internet Information Server.*

- *MS Transaction Server Component includes the header files needed by MTS, and makes the object non-aggregatable.*
- *Component Registrar Object adds an object that implements the IComponentRegistrar interface, which is used to register the objects in a server individually.*
- **Controls:**
 - *Full Control adds an object that implements all the interfaces needed by full ActiveX control (OCX), such that it should work with any ActiveX Control Container.*
 - *Internet Explorer Control adds a control that has all the interfaces (and UI functionality) needed by Internet Explorer and other compatible browsers.*
 - *Property Page adds a property page object.*
- **Miscellaneous:**
 - *Dialog adds a class that implements a dialog*

-
- 1.3.1.2.2 Description of the available options for generating the code (parameters, code-templates etc.)
(1.3.1.2.2) It is easy to alter the generation. You can choose many different setting. The following gives an overview of the settings:
 - *Names: Classnames, Filenames, CoClass name, Interface name, Prog ID*
 - *Threading Model: Single, Apartment, Both, Free*
 - *Interface: Dual, Custom*
 - *Agregation: Yes, No, Only*
 - *Support for error info*
 - *Support for Connection points*
 - *Using the Free Threaded Marshaler*

-
- 1.3.1.2.3 Description of the architecture of the generated code ?
(1.3.1.2.3) The generated code is pretty complex. It contains files for both creating the COM objects and the typelibrary. The Typelibrary is defined in the .idl file and the registration of the components is handled in the .reg files. For each COM class you create you will get a .h and a .cpp where the specific code for the component is defined (This is here you are going to code). The Project also contains files for definition of the objects in the system.

-
- 1.3.1.2.4 Is it easy to customize the generated code ?
(1.3.1.2.4) The generated code can easily be altered (maybe to easy). The many macros defines the objects and if they are altered the behaviour is changed.
-

- 1.3.1.2.5 Do the wizards allow for reverse-engineering ? To answer this question you should answer the following
(1.3.1.2.5) Generating the code is a one way thing. If you regret anything a new project has to be started. Adding objects and methods is also a forward only thing.

-
- 1.3.1.2.5.1 Are customizations in the generated code preserved when reverse-engineering the code ?
(1.3.1.2.5.1) Reverse engineering is not possible. But you can make additions to the code using the wizards.

-
- 1.3.1.2.5.2 Is it possible to change the fundamental strategies for the generated code ?
(1.3.1.2.5.2) The strategy for (the outlook of) code is set.

-
- 1.3.1.2.5.3 To evaluate the maintenance of the generated code (customizations, additions, changes, etc.): What changes are allowed in the generated code while still retaining the possibility of reverse-engineering it ?
(1.3.1.2.5.3) Reverse engineering is not possible. Any changes to methods and parameters has to be done by hand (at least three different places (idl, .h, .cpp).

-
- 1.3.1.2.6 Evaluation of the quality of the generated code
(1.3.1.2.6) The generated code performs very well (this is the essence of ATL).

-
- 1.3.1.2.6.1 How is the performance, stability and coding standard of the generated code ?
(1.3.1.2.6.1) The code is very compact and very fast.

-
- 1.3.1.2.6.2 Comments:

- 1.3.2 Component creation using handcoding
(1.3.2) The task of generating the components by hand is possible but a stupid task to begin because it is a bit complicated.

-
- 1.3.2.1 Is it easy?
(1.3.2.1) No

1.4 Support for interfaces

(1.4)

-
- 1.4.1 Which types of interfaces do the tool support:
(1.4.1)

-
- 1.4.1.1 Custom/vtable ?
(1.4.1.1) Yes, Pure Custom/VTable interfaces are supported directly.
-

- 1.4.1.2 Dispatch ?
(1.4.1.2) Yes, Pure Dispatch interfaces are supported, but you will have to manually delete the vtable interface because you as default when generating a new interface you can only choose between custom (vtable) and dual interfaces.
-
- 1.4.1.3 Dual ?
(1.4.1.3) Yes, Dual interfaces are supported.
-
- 1.4.2 How do you define an interface?
(1.4.2) Creating an interface can be done in various ways. If you use the wizard you will automatically get at least one interface created at the same time as the object (coclass). Depending on the wizard you can create either one or many interfaces during creation. The "New ATL Object.." wizard will only let you create one interface during creation. The "New Class.." wizard will let you create multiple interfaces during creation.
-
- 1.4.3 Is it possible to implement multiple interfaces on a component ?
(1.4.3) Yes, either manually or using the wizard during object creation.
-
- 1.4.3.1 If Yes, how ?
(1.4.3.1) Using the wizard see 1.4.2. Manually you first define the interface in the IDL. The definition tells whether it is a custom (Inherited from IUnknown), a dual (inherited from IDispatch, with the dual keyword in the header), and a pure dispinterface by using the keyword "dispinterface" instead of the normal keyword "interface". Then you add the interface to the coclass. If it is a custom, or a dual interface you use the keyword "interface" in the coclass definition and if it is a pure dispinterface you use the keyword "dispinterface" in the coclass. The following defines an coclass with three different interface (a custom interface, a dual, and pure dispinterface):

```
[ uuid(02B6FBB1-F794-11d0-BFFF-00A0C922E84A),  
helpstring("Dispatch interface for Clock")  
]  
dispinterface IClock  
{  
properties:  
[id(1)] short AlarmHour;  
[id(2)] short AlarmMinute;  
methods:  
[id(3)] void TestAlarm();  
};  
[ object,  
uuid(37FBCAE0-08B6-11d1-8001-00A0C922E84A),  
dual,  
helpstring("IArc Interface"),  
pointer_default(unique)  
] interface IArc : IDispatch  
{  
[id(1), helpstring("method ArcSin")] HRESULT ArcSin([in]  
double arg, [out, retval] double * result);  
[id(2), helpstring("method ArcCos")] HRESULT ArcCos([in]  
double arg, [out, retval] double * result);
```

```
[id(3), helpstring("method ArcTan")] HRESULT ArcTan([in]  
double arg, [out, retval] double * result);  
};  
[ uuid(64711D31-08C1-11D1-98AB-00A0D100E3C8),  
helpstring("IAmACustomInterface Interface"),  
pointer_default(unique)  
]  
interface helpstring("IAmACustomInterface : IUnknown  
{...  
};  
[ uuid(02B6FBB2-F794-11d0-BFFF-00A0C922E84A),  
helpstring("CoClass for clock")  
]  
coclass Clock  
{  
[default] dispinterface IClock;  
interface IArc;  
interface IAmACustomInterface;  
};
```

You then add the interfaces to the inheritance list for the com object (in the true .h file), either directly, or using then macro IDispacthImpl, both After doing that interface should be added to the interface map (the stuff between the BEGIN_COM_MAP and the END_COM_MAP macros using the macro COM_INTERFACE_ENTRY. The rest is just to define the methods in the .h-file and implement them in the .cpp file.

- 1.4.4 Predefined interfaces:

(1.4.4) Yes, predefined interfaces can easily be implemented.

-
- 1.4.4.1 Is it possible to implement predefined interfaces, i.e. implementing interfaces defined by other people (preserving IID's)?
(1.4.4.1) It is easy to implement other interfaces defined by other people. You just include the interface in the idl-file (you can usually clip it from the object viewer) and follow the steps in 1.4.3.1.

-
- 1.4.4.2 What do you use as foundation for the predefined interface ?
(1.4.4.2) The whole task is done in the IDL. The questions is just how you get the right IDL code? This will be described in the following

- 1.4.4.2.1 Type lib ?

(1.4.4.2.1) The interface definition can be achieved from the type library of another component that implements the interface. Using the tool "Object Viewer" that comes with VC++ 5.0 the IDL code for the typelibrary can be cut and pasted into your own project. It is now up to the programmer to implement the interface manually in the right manner.

- 1.4.4.2.2 IDL ?

(1.4.4.2.2) The interface definition can either be typed in manually in the IDL code or achieved as in 1.4.4.2.1

- 1.4.4.2.3 Source code ?
(1.4.4.2.3) No, using the C++ source code only is not enough because the interface have to be defined too.
-

- 1.4.4.2.4 other
(1.4.4.2.4) No, no other way is known.
-

- 1.4.4.3 How do you use a predefined interface?
 - 1.4.5 Interface maintenance
(1.4.5) In the following interface maintenance is described
-

- 1.4.5.1 Which types of changes in a published interface are allowed ?
(1.4.5.1) New methods can be added using a wizard or doing it manually. Adding extra parameters after the wizard is used has to be done manually in the IDL-file, the .h-file and the implementation file (the .cpp-file). This is a bit tedious.
-

- 1.4.5.1 Insert new methods and/or parameters ?
(1.4.5.1) It is always possible to change an interface (both published and non-published) eventhough it is against the rules of COM.
-

- 1.4.5.2 Change existing methods and/or parameters ?
(1.4.5.2) Changing has to be done manually all three places as mentioned in 1.4.5.1.
-

- 1.4.5.3 Delete methods and/or parameters ?
(1.4.5.3) Deleting also has to be done manually.
-

- 1.4.5.4 What are the consequences of changes on the generated component ?
(1.4.5.4) If the changes aren't done in all the places (idl, .h, and .cpp) the compiler will choack on the interface and the implementation.
-

- 1.4.6 Interface optimizations:
(1.4.6) Some interface optimization can be done, see 1.4.6.2
-

- Which type of interface optimization are supported:
 - 1.4.6.1 None?
(1.4.6.1) No, some optimization can be done
-

- 1.4.6.2 Tear-of interfaces (a tear-of interface is part of an aggregated component. The component will not be instantiated until the interface is actually queried), How: ?
(1.4.6.2) Tear-off interfaces can be made using the superclass CComTearOffObjectBase and the macro COM_INTERFACE_ENTRY_TEAR_OFF in the COM map for

the object. Remember that tear-offs is an optimization of aggregated objects. There is an optimization for the tear-offs. This is COM_INTERFACE_ENTRY_CACHED_TEAR_OFF where the inner objects IUnknown is cached, so when first instantiated it is cached for further usage.

- 1.4.6.3 Other:
(1.4.6.3) No other optimizations are known.
-

1.5 Reference counting *(1.5) Svartekst*

- 1.5.1 A description of aspects of reference counting
(1.5.1) Svartekst
-

- 1.5.1.1 This question is intentionally left blank
(1.5.1.1) Svartekst
-

- 1.5.1.2 Is reference counting automatically supported for aggregated components?
(1.5.1.2) Yes.
-

- 1.5.2 How is reference counting implemented in the tool ?
(1.5.2) Svartekst
-

- 1.5.2.1 Is AddRef called automatically by a generated component when returning an interface reference ?
(1.5.2.1) When getting an interface from the component, like from QueryInterface, ATL will automatically obey the rules of COM that an outgoing interface should AddRef on the interface. If the method on the component is created by you self it is your own responsibility to call AddRef on the interface.
-

- 1.5.2.2 Is Release called automatically by a generated component when exiting scope for an interface reference received as a parameter ?
*(1.5.2.2) In VC++ interfaces can be used in at least three different ways:
(1) Pure C++ (interface) pointers. Result: Release must be called manually and this can be errorprone.
(2) Pure VC++ v5.0 smartpointers (_com_ptr_t). Result: Release is called automatically when running out of scope.
(3) Pure ATL smartpointers (...). Result: Release is called automatically when running out of scope.
The difference between (2) and (3) is that (2) uses the C-Runtime library which increases the size of the component.*
-

- 1.5.2.3 What happens when the reference count drops to 0 ?
(1.5.2.3) The default implementation is to delete the object.
-

- 1.5.2.3.1 Is it possible to customize what happens when the reference count drops to 0 ?
(1.5.2.3.1) The default implementation is add to the object by inheriting from CComObjectRootEx. If you manually delete this superclass from the inheritance list and you manually implement the AddRef and Release methods you can customize the behaviour.

-
- 1.5.2.4 How does the tools mechanism for implementing reference counting affect flexibility etc. ?
(1.5.2.4) It doesn't influence on flexibility, but it can be difficult to implement your own strategy.

1.6 Threading *(1.6) Svartekst*

-
- 1.6.1 Which threading models are supported ?
(1.6.1) Svartekst
 - 1.6.1.2 Not specified ?
(1.6.1.2) -
 - 1.6.1.3 Single ?
(1.6.1.3) Single threading is supported.
 - 1.6.1.4 Apartment ?
(1.6.1.4) Apartment threading is supported.
 - 1.6.1.5 Free ?
(1.6.1.5) Free threading is supported.
 - 1.6.1.6 Other:
(1.6.1.6) No other threading models is supported
 - 1.6.2 Multi threaded components:
(1.6.2) Multithreading has to manually implemented using the C-Runtime methods `_begin_thread` and the multithreaded runtime libraries. There are support for critical sections.
 - 1.6.2.1 Can a component be multi-threaded ?
(1.6.2.1) Yes, a component can be multithreaded
 - 1.6.2.2 If no, is multi-threading supported through language facilities (i.e. in native java) or is it near API ?
(1.6.2.2) There are no support classes for threading objects as in MFC. These can easily be implemented by hand.

1.7 Throwing errors

(1.7) Svartekst

- 1.7.1 Do generated components support:
(1.7.1) It is an option whether or not the objects should support any form of COM standard error notification.

 - 1.7.1.1 ISupportErrorInfo ?
(1.7.1.1) Yes, ISupportError is supported when creating objects using the wizards. The support is through the template class ISupportErrorInfoImpl.

 - 1.7.1.2 this question is intentionally left blank
(1.7.1.2) No, IObjectError is not supported.

- 1.7.2 Direct support for HRESULT
(1.7.2) Error handling can be done either through direct HRESULT, or through VC++ native COM exceptions.

 - 1.7.2.1 Does the tool support the use of existing HRESULT's ?
(1.7.2.1) Yes.

 - 1.7.2.2 Does the tool support the use of interface dependent HRESULTS ?
(1.7.2.2) Yes, if they are defined somewhere.

 - 1.7.2.3 Does the tool provide support for HRESULT creation ?
(1.7.2.3) Yes.

 - 1.7.2.4 Is it easy to create/use HRESULTS ?
(1.7.2.4) As easy as the concept of HRESULTS permits.

- 1.7.3 Are there mappings between HRESULTS and native language exceptions ?
(1.7.3) If the #import statement is used the HRESULT is automatically translated into _com_error exceptions.

 - 1.7.3.1 Is the mapping manual ?
(1.7.3.1) See 1.7.3.

 - 1.7.3.2 Is the mapping automatic ?
(1.7.3.2) See 1.7.3.

- 1.7.4 How does the mapping work ?
(1.7.4) See 1.7.3.

1.8 Datatypes

(1.8)

- 1.8.1 How are data-types of the development tools mapped to COM data types ?
(1.8.1)

-
- 1.8.1.1 Simple datatypes ?

(1.8.1.1) The usual automation types have the following direct mapping: Some of the types can be mapped into some smarter C++ types: VT_DISPATCH and VT_UNKNOWN can be mapped into _com_ptr_t smartpointers or the ATL pendang. For strings see underneath

-
- 1.8.1.2 Strings ?

(1.8.1.2) Strings are mapped as follows: VT_BSTR BSTR or _bstr_t VT_LPSTR LPSTR or _bstr_t VT_LPWSTR LPWSTR or _bstr_t There are also predefined macros to do conversion between the different types. See ATL materialet

-
- 1.8.1.3 Predefined types (automation compliant types)?
(1.8.1.3) ?

-
- 1.8.1.4 User-defined types (e.g. struct, enum)?

(1.8.1.4) Both structs and enums are mapped directly

-
- 1.8.1.5 Object parameters (interface references) ?

(1.8.1.5) Interface references are mapped into native interface pointers.

-
- 1.8.1.6 Variants ?

(1.8.1.6) Variants are mapped into VARIANTS or the ATL type CComVariant

-
- 1.8.1.7 SafeArrays ?

(1.8.1.7) SafeArrays have to be manipulated through the usual API methods and the SAFEARRAY.

1.9 Marshalling

(1.9)

-
- 1.9.1 How can marshalling be handled:
(1.9.1)

-
- 1.9.1.1 Don't know! ?

(1.9.1.1) -

-
- 1.9.1.2 OLE32.DLL ?

(1.9.1.2) The normal Dispatch interfaces are marshalled through OLE32.DLL and OLEAUT32.DLL.

-
- 1.9.1.3 Own Proxy/Stub DLL ?

(1.9.1.3) A projects for creating a proxy/stub dll is automatically generated as a side effect of creating a normal project.

-
- 1.9.1.3.1 Can it be attached to the component DLL?
(1.9.1.3.1) *No, not that I know of. It is also smarter for the proxy DLL not to be.*

-
- 1.9.1.4 Directly made by tool ?
(1.9.1.4) *Yes, it is made by the tool.*

-
- 1.9.1.5 Work around ?
(1.9.1.5) *No*

-
- 1.9.2 Does the tool support IMarshal ?
(1.9.2) *When creating your own IMarshal interface you are on your own, but*

1.10 How does the server support typelibraries (1.10)

-
- 1.10.1 IDL Integration ?
(1.10.1) *Yes, through IDL integration*

-
- 1.10.1.1 IDL/ODL ?
(1.10.1.1) *Yes, through IDL integration*

-
- 1.10.2 Implicit creation of typelibraries ?
(1.10.2) *The type lib is created during normal compilation by calling the MIDL compiler.*

-
- 1.10.3 Integration of typelibraries as ressource in server ?
(1.10.3) *The typelibrary can be attached to the server or it can be distributed seperatly.*

-
- 1.10.4 Are errors in the type library able to "cheat" the tools ?
(1.10.4) *The compiler should catch the errors.*
-

2. Compilation and Distribution

2.1 Is version compatibility checking performed during compilation and how ? (2.1)

2.2 Interface compatibility (2.2)

-
- 2.2.1 Is it possible to changes an interface and keep the same IID?
(2.2.1) *Yes, you can do everything to an interface without changing the IID.*
-

2.3 Which types of target code does the tool produce (i.e. native/optimized native + runtime, p-code, nothing/interpreted) ?

(2.3) There are many different types of targets, but they are all compiled native Win32 code. The different types are: Debug Unicode Debug Release Minimum size Release Minimum Dependency The ATL registrar (registrar = registry resource interpreter) Unicode Release Minimum size Unicode Minimum Dependency

2.4 How does the tool support registration of components ?

(2.4) The registry code is included as a resource for the component. This code can be interpreted by the ATL registrar. This registrar can be either linked to the component or it can be called from the ATL.DLL that should be distributed with the component.

- 2.4.1 Components are selfregistering ?

(2.4.1) All components are (and should be) selfregistering

- If Yes then

- 2.4.1.1 DLL RegSvr32 ?

(2.4.1.1) On DLL server: The RegSvr32

- 2.4.1.2 EXE: selfregistering when run once ?

(2.4.1.2) EXE: Selfregistering when ran onced or ran with the switch /register. Unregistering is done by calling the exe-file with the switch /unregister.

- 2.4.1.3 EXE: selfregistering when compiled ?

(2.4.1.3) No, you will have to do the registration manually.

- 2.4.2 Registration is done through

(2.4.2)

- 2.4.2.1 Installation ?

(2.4.2.1) The components can be registered directly through installing by using REG-scripts, by this is a very errorprone approach.

- 2.4.2.2 Has to be done manually ?

(2.4.2.2) The registration should be done by using selfregistration as desriped in 2.4.1.1 and 2.4.1.2.

- 2.4.3 Does the tool support unregistration of components ?

(2.4.3) Unregistration is done either through RegSvr32 for DLL's or the /unregister commandline switch for EXE-servers.

- 2.4.4 Does the tool support registration using ProgID ?

(2.4.4) Yes, registration using ProgID's can be done by specifying it through object creation (wizard) or manipulation the .rgs-file for the project.

- 2.4.4.1 Can the developer freely choose a components ProgID ?
(2.4.4.1) *Yes, you can freely choose a components ProgID.*

-
- 2.4.5 Does the tool support version independent ProgID's, and if yes how ?
(2.4.5) *Version independant ProgID's is supported through the registrar scripts (the .rgs script for the project). In the script you write something like:*

```
MyServer.MyComponent = s 'MyServer Class'  
{  
  CurVer = s 'MyServer.MyComponent.1'  
}
```

2.5 Support for component categories: (2.5) ?

-
- 2.5.1 Is it possible to define and implement your own component categories ?
(2.5.1) ?
 - 2.5.1.1 How ?
(2.5.1.1) ?

-
- 2.5.2 Does the tool have support for predefined component categories and how ?
- ## 2.6 Can the developer freely choose CLSID for a component ? (2.6) *Yes, a developer can freely choose CLSID for a component. The GUID for the component can be generated by using the tools guidgen.exe or genguid.exe.*

3. Component Client

3.1 Support for COM clients

3.1.1 Creating and deleting components (3.1.1)

-
- 3.1.1.1 Is it easy to create new components ?
(3.1.1.1) *Creation of components can be done either by using one of the wizards or by hand. The first approach is preferable for the inexperinced programmer because the it is very error prone.*
 - 3.1.1.2 Is it easy to delete existing components ?
(3.1.1.2) *This has to be done manually and includes deleting from the .IDL-file, the project-cpp file, and delting the definition (.h) and implementation (.cpp) for the object. You should also delete the registration code from the .rgs-file. So Yes it is easy if you do it right and difficault if you don't know how to do it!*
 - 3.1.1.3 Does the tool delete registry entries for deleted components after recompilation ?
(3.1.1.3) *No, you will have to manually delete the entries from the registry.*
-

- 3.1.1.4 This question is intentionally left blank
(3.1.1.4) *Yes*

3.1.2 COM Library Support (3.1.2)

- 3.1.2.1 Does the tool allow direct access to COM functionality, like calling COM functions direct (CoCreateInstance, CreateInstance, ...). ?
(3.1.2.1) *Off course, this is C++ where API calls are a must.*
- 3.1.2.2 Does the tool encapsulate COM functionality (indirect COM support) ?
(3.1.2.2) *There are two approaches for indirect COM-support. The new VC++ v5.0 #import statement makes you include type libraries into your project. If you want you can use the `_com_ptr_t` COM-interface-smartpointer. You can also use the ATL pendant `CComPtr` template. The #import statement dynamically generates some files that is included in the project. These files are the definitions for the typelibrary that is imported so it can be used in native C++.*
- 3.1.2.3 Does the tool provide COM Helper Classes ?
(3.1.2.3) *Yes, some helper classes are supported: `_com_ptr_t`, `_bstr_t`, `CComPtr` are the most used.*
- 3.1.2.4 Is it possible to combine native and encapsulated COM support ?
(3.1.2.4) *Yes, native and encapsulated Com support can be combined.*

3.1.3 Acquisition of interfaces (3.1.3)

- 3.1.3.1 Is it possible to call QueryInterface directly ?
(3.1.3.1) *Yes, you can call `QueryInterface` directly.*
- 3.1.3.2 Does the tool encapsulate QueryInterface ?
(3.1.3.2) *Yes, the tool can encapsulate the `QI`-calls through the COM-smartpointers.*
- 3.1.3.3 Reference counting ?
(3.1.3.3)
 - 3.1.3.3.1 Is AddRef called automatically by a generated client when using an interface as parameter for a (server) method ?
(3.1.3.3.1) *When using normal C++ interface pointers the AddRef should be called manually. When using a smartpointer this is done.....(?)*
 - 3.1.3.3.2 Is Release called automatically by a generated client when exiting scope for an interface reference ?
(3.1.3.3.2) *When using normal C++ interface pointers the Release should be called manually. When using a smartpointer this is done automatically.*

3.1.4 Component binding (3.1.4)

- 3.1.4.1 Is early binding supported through: ?
(3.1.4.1) *The binding to component is easiest done through the #import statement.*
 - 3.1.4.1.1 VTable binding ?
(3.1.4.1.1) *For interfaces that is dual a vtable binding is performed. If you want to bind through the Dispatch interface you will have to do that manually for dual interface.*
 - 3.1.4.1.2 DispID binding ?
(3.1.4.1.2) *For pure dispatch interfaces (no vtable) the DispID binding is default when using the #import statement. If the interface is dual the binding is allways on the vtable. If you in that case want to do DispID (with known DispID) binding it has to be done through explicit calls to Invoke.*
 - 3.1.4.2 Is late binding supported and how ?
(3.1.4.2) *Late binding can be achieved by explicitly calling GetIDsOfNames and Invoke, but it is ugly code.*
 - 3.1.4.3 Binding through ProgID
 - 3.1.4.3.1 Does the tool support binding of (server) component through version independent ProgID ?
 - 3.1.4.3.2 Does the tool support binding of (server) component through version specific ProgID ?
-

3.1.5 Is the process of catching errors supported through:

(3.1.5) *COM error handling can be done with the #import statement where HRESULT's are translated into native C++ exceptions (_com_error_t). They can also be handled as normal HRESULT's and deciffered by using macros like SUCCEEDED(hres) and FAILED(hres).*

- 3.1.5.1 native COM support (i.e. using HRESULT) ?
(3.1.5.1) *Yes*
 - 3.1.5.2 encapsulated support (in the tool language/environment/library) ?
(3.1.5.2) *Yes*
-

3.2 Aggregation

3.2.1 Can the tool control COM-aggregation for produced components ?

(3.2.1) *When creating in-proc components it is a choise in the wizard to choose between different kinds of aggregation. The choises are:*

(1) *Aggregatable.*

(2) *Never aggregatable. The component can not be instantiated using a outer unknown*

interface

(3) Only aggregatable. Methods like CoCreateInstance can for this kind of objects only be called with a known outer interface. When trying to make explicit instances of these components the CoCreateInstance will fail.

(4) Poly aggregatable. The object is created from the same object whether or not it is aggregated.

-
- If yes, Supported methods
 - 3.2.1.1 Can you specify that a component is not aggregatable ?
(3.2.1.1) Yes
-
- 3.2.1.2 Is the aggregation support Wizard oriented (GUI) ?
(3.2.1.2) Yes, but it can also be done by hand
-
- 3.2.1.3 Is Manual programming of aggregation Easy/Difficult ?
(3.2.1.3) When doing manual programming the macro DECLARE_xxxAGGREGATABLE, where (xxx can be substituted with nothing, "NOT", "ONLY", and "POLY". The DECLARE_AGGREGATABLE is default. These macros should be added in the public part of the .h file for the component. So it is very easy to control the aggregation behaviour for the component, but it has to be done by hand.
-

3.3 Delegation

3.3.1 Support for delegation is:

(3.3.1)

-
- 3.3.1.1 Manual ?
(3.3.1.1) Support for delegation has to be done manually. You will first have to define the interface for the component in the IDL file. In the .h file you must create an interface pointer to the inner component. This component can be initialized in the FinalConstruct virtual method and Released in the FinalDestruct virtual method. All methods from the interface should be delegated to the inner component and this has to be done by the component programmer.
-
- 3.3.1.2 Tool ?
(3.3.1.2) No
-

4. Development Environment support

4.1 Description and evaluation of supporting tools/facilities in the development tool

(4.1)

- 4.1.1 What kind of object browser facilities are provided ?
(4.1.1) For browsing typelibraries the VC++ v5.0 comes with a tool called "OLE-COM object viewer". This tool can browse typelibraries from files, and it can read the registry to find different component categories.

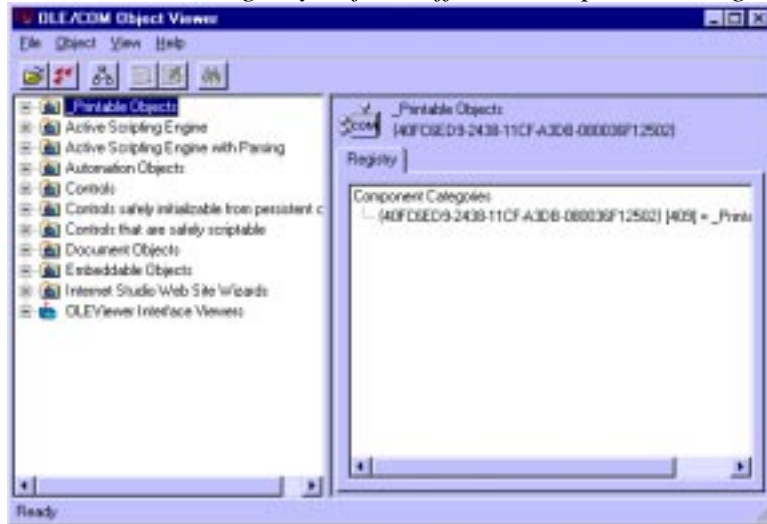


Figure X: VC++ 5.0 OLE-COM Object Viewer

From the environment (the IDE) the projects different components can be viewed through the class view:

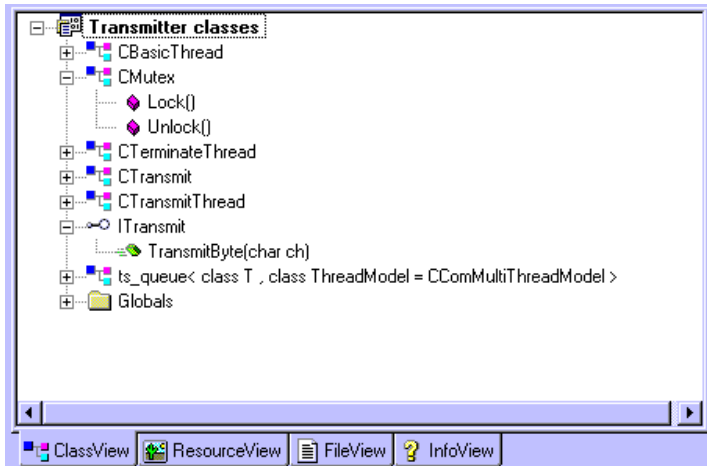


Figure X: VC++ 5.0 Class View with COM interfaces

-
- **4.1.2** What kind of syntax help/info for COM-components and methods herein ?
(4.1.2) You can not get the intellisense as you have in Visual Basic 5.0, so COM programming is made easier. You will have to know the method names and parameters in advance or the compiler will vomate. This will be made in next editions of VC++.
-
- **4.1.3** Does the tool provide context sensitive help for components (to be used by component integraters) ?
*(4.1.3) In the IDL file you can specify a Help Key number to be used by the component user. The help file has to be created manually. You can also give a direct help by using the "helpstring" keyword in the IDL:
[id(1), helpstring("method TransmitByte")] HRESULT
TransmitByte([in] char ch);*
-
- **4.1.4** Support for interface prototypes (components implementing interfaces specified externally)
(4.1.4) VC++ and ATL provides default implementations for the interfaces: IUnknown, IDispatch, ISupportErrorInfo, and a lots of interfaces for creating different kinds of controls (IE controls, visual controls, etc.).
-

- 4.1.4.1 How can you make this manually ?
(4.1.4.1) This can be done as if the interface was your own to implement.
-

- 4.1.4.2 What kind of toolsupport is provided (the tool provides prototypes for the methods of the interface). Filling out the slots ?
(4.1.4.2) The usual implementation is hidden in the superclasses, but as with most virtual methods these can be overwritten and customized in your class. You cannot make the tool create slots for predefined interfaces created by others and then let you fill in the gaps.
-

- 4.1.5 Debugging
(4.1.5)
-

- 4.1.5.1 Which facilities exist for debugging COM-components ?
(4.1.5.1) Debugging COM components is as debugging normal C++ code. If the component is an in-proc server, You just specify the name of the EXE file that uses the component. If the component is an ActiveX control with the interfaces corresponding to an OCX the ActiveX Test container can server the purpose of the EXE file. If the server is an out-proc server You can start it from the IDE with the command line switch "/Embedding" and when another program, like a VB program uses the server the debugger will be woken up (if you have set a breakpoint).
-

- 4.1.5.2 Is it possible in the development to debug from client into component ?
(4.1.5.2) Yes, see 4.1.5.1.
-

- 4.1.5.3 If any, what are the differences between debugging in-process and out-of-process servers ?
(4.1.5.3) Yes, see 4.1.5.1.
-

4.2 Evaluation of the development tool *(4.2)*

- 4.2.1 A description of the work load
(4.2.1)
-

- 4.2.1.1 Does the COM-based part of the development integrate nicely and consistently into the tool ?
(4.2.1.1) When the concept of COM is known by the programmer it is pretty easy to develop COM components. There are though problems when going from Debug versions to Release versions. As default ATL assumes you don't use the C-Runtime library for the release versions of the components. This can for an inexperienced ATL programmer that is about to ship his component, give a lot of headache. If you use the C- Runtime library (i.e. by using the #import directive) you should delete the symbol _ATL_MIN_CRT from the defines-list.
-

- 4.2.1.1.1 Is it easy to find out *how* to do COM-based development in the tool ?
(4.2.1.1.1) The VC++ compiler comes with some good tutorials for creating servers and ActiveX controls. The concept of COM must be known.

- 4.2.1.1.2 Is the COM-based development in the tool similar to non-COM-based development ?
(4.2.1.1.2) ATL can also be used for creating dialog based programs, but the rest of ATL is COM. The usual MFC code can be integrated into the framework, but then you loose the idea of having small and efficient components.

- 4.2.1.2 Is it easy for novices/experts with respect to the COM-development in general/the development tool specifically to do COM-based development in the tool.
(4.2.1.2) Novices to COM should read a book about COM before using ATL because some knowledge to COM is assumed. The greatest advantage of ATL is that it is flexible. The component can be "revet og flået i" until it fits your task. If it fails to do the COM task you wanted you can always turn back to "pure" COM (through COM API methods) within the same component. You are never stuck with a specific implementation for a method in a standard interface. If it doesn't suit you, you can change the implementation by copying the old code and create you own implementation. This latest flexiblity is due to the fact that the framework is based on source code that is delivered with ATL.

5. Development Processes

- 5.1 Is it possible to develop and test at the same time ?
(5.1) Yes

- 5.1.1 Will recompilation require change of server GUID and IID's ?
(5.1.1) No
