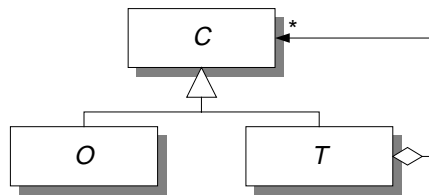


# *Evaluation of COM Support in Delphi*

*COT/3-5-V1.0*



Centre for Object Technology

Revision history: V1.0 18-08-98 First version.

Author(s): Peter Petersen, Aarhus University  
Kåre Kjelstrøm, Aarhus University

Status: Complete

Publication: Public

Summary:

This document presents the answer to a number of specific questions regarding the support for COM based development using Borland Delphi.

© Copyright 1998 Aarhus University.

## Introduction

This document presents the answer to a number of specific questions regarding the support for COM based development using Borland Delphi.

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) You can make ActiveX controls from all native Delphi visual controls i.e. buttons, forms, grids etc. Delphi will automatically provide a skeleton file in which you will simply fill out the implementations of properties and methods.*

---

- 1.1.1 Support for visual Controls ?

*(1.1.1) You can make ActiveX controls from all native Delphi visual controls i.e. buttons, forms, grids etc. Delphi will automatically provide a skeleton file in which you will simply fill out the implementations of properties and methods.*

---

- 1.1.2 Support for non-visual Controls ?

*(1.1.2) All kinds of COM components from IUnknown to IDispatch based ones can be made.*

---

1.2 Types of servers

*(1.2.1) Yes*

---

- 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) *No*
- 

- 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) *No*
- 

- 1.2.3.3 Restarting the project and moving the code manually ?  
(1.2.3.3) *You must create a new project to make the transition. Delphi supports a wizard that will initially let you specify the kind of server you want to create. To change the type you will start a new project of that kind and then include the implementation units from the old project.*
- 

- 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) *No*
- 

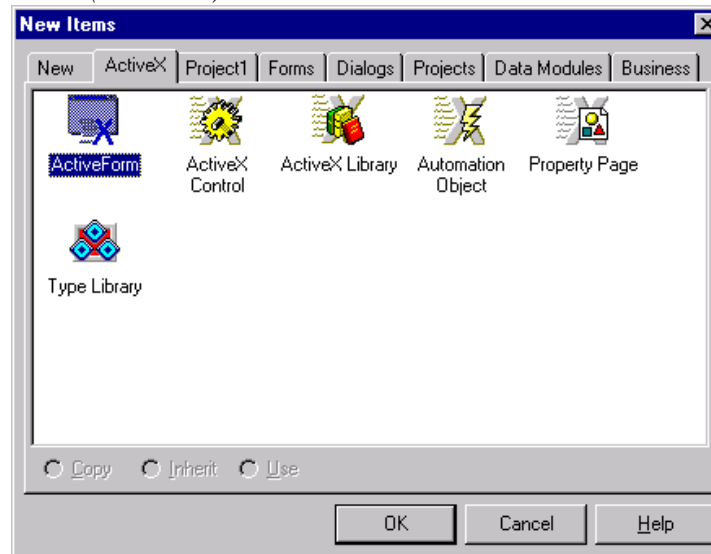
### 1.3 Component construction (tool support)

- 1.3.1 How are the COM components created ?  
(1.3.1.1) *No*
-

- 1.3.1.1 Not using wizard. Comments:  
(1.3.1.1) No
- 

- 1.3.1.2 using wizard. Comments:  
(1.3.1.2) Yes
- 

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



**Figure 1: Delphi COM Wizards**

*Delphi has seven wizards that you must acquaint yourself with, when you want to start making components. Six of these are shown on Figure 1, and the seventh is simply the ordinary executable wizard.*

*To choose between in-process and out-of-process you will select either*

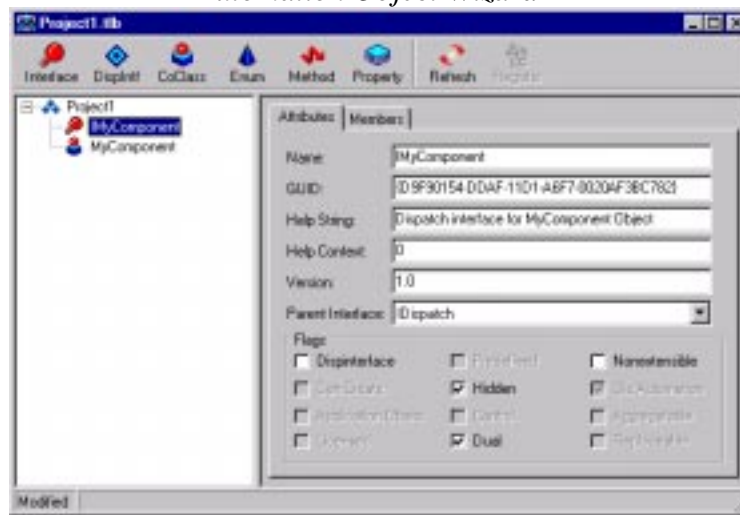
- **Application:** Starts a new empty project for an ordinary executable (EXE)
- **ActiveX Library:** Starts a new empty project for a library (DLL). The name of the wizard is misleading in that simple COM servers that do not implement IDispatch or any ActiveX specific interface will also be created this way.

*When a new application type has been defined you will then select a new component type. This can be one of the following:*

- **Automation Object:** A simple COM component that can derive from IUnknown, IDispatch, and a number of others including the ones that you have defined yourself. Following the definition of the name of your new

component, the wizard will open the Type Library Editor as shown on Figure 2 and insert an interface and CoClass definition for you.

- **ActiveForm:** An ActiveX control that represents an entire visual form, on which you can drop native Delphi controls. In all other respects, this is an ActiveX control.
- **ActiveX Control:** ActiveX control that derives from a native Delphi control other than a form. This can be a button, a grid etc.
- **Property Page:** A Delphi wrapper for the property pages technology
- **Type Library:** Opens the Type Library Editor with an empty library. You will typically not use this wizard when creating component, but rather define them through the Automation Object Wizard



**Figure 2: Type Library Editor**

- 1.3.1.2.2 Description of the available options for generating the code (parameters, code-templates etc.)  
*(1.3.1.2.2) When you first create a new component using the Automation Object wizard, you will be asked for the name of the new control and how it should be allowed to be instantiated. The possible choices are single, multiple or internal. Internal specifies that no external application is allowed to use the component, whereas single and multiple are self-explanatory. The Type Library Editor will be opened next, and from here you may set all kinds of properties on the component. These correspond to flags and attributes in IDL. There does not seem to be any restrictions on the attributes, but you can only create methods, properties, interface, CoClass and enumerations from the editor. Struct, unions and modules can be shown by opening an existing type library, but not added.*

- 1.3.1.2.3 Description of the architecture of the generated code ?  
*(1.3.1.2.3) Once the library is fully defined, you will register it on the button in the upper right corner of the Type Library Editor. This will create the necessary entries in the Windows Registry and create a number of source files. Each CoClass will get its own unit, which contains skeleton source for all methods and properties in the implemented interfaces as well as initialisation code for a corresponding Class Factory. Furthermore, a file containing pascal definitions of CoClass GUIDS, Interfaces, constants, structures and unions. This file has "\_TLB" appended to its name.*
- 

- 1.3.1.2.4 Is it easy to customize the generated code ?  
*(1.3.1.2.4) You cannot alter the \_TLB file, since this will be auto-generated each time the type library changes via the editor. All other code can be customised as you see fit.*
- 

- 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) If you redefine method or property definitions in the Type Library Editor following an implementation of some of them, your code will not be harmed.*
- 

- 1.3.1.2.5.1 Are customizations in the generated code preserved when reverse-engineering the code ?  
*(1.3.1.2.5.1) Yes*
- 

- 1.3.1.2.5.2 Is it possible to change the fundamental strategies for the generated code ?  
*(1.3.1.2.5.2) No. Delphi has its own way.*
- 

- 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) *The only file you cannot customise to your own preferences is the `_TLB` type library wrapper file.*

---

- 1.3.1.2.6 Evaluation of the quality of the generated code  
(1.3.1.2.6) *Overall the generated code is ok, and performs fine. There is a problem, though.*
- 

- 1.3.1.2.6.1 How is the performance, stability and coding standard of the generated code ?  
(1.3.1.2.6.1) *Sometimes the Type Library Editor changes its definitions of methods and properties in the `_TLB` file. The one minute you will have code that looks like:*  

```
function MyFunc(V1: Integer): Boolean;
```

*After a refresh, however you may end up with something that looks like:*  

```
function MyFunc(V1: Integer; var V2: Boolean): HRESULT;
```

*The CoClass skeleton declarations are not updated, and so you end up with a project that cannot compile unless you manually redefine the CoClass skeleton files. We have not been able to pinpoint the situation in which this problem occurs.*
- 

- 1.3.1.2.6.2 Comments:  
(1.3.1.2.6.2) *The Type Library editor has a number of known bugs that will eventually be fixed by Borland. A report is available from [Borland's homepage](#)*
- 

- 1.3.2 Component creation using handcoding  
(1.3.2) *If you want to define the entire project yourself, there is some work to do, but it is not impossible. You will have to define the project file, the CoClass skeleton files and the type library by hand, IDL, or Type Library Editor. [The CASE Tool IDUN](#) autogenerates these files from a type library description.*
- 

- 1.3.2.1 Is it easy?  
(1.3.2.1) *It is not worth the while, but can be done.*
-

## 1.4 Support for interfaces

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

- 1.4.1.1 Custom/vtable ?  
*(1.4.1.1) Yes*
- 

- 1.4.1.2 Dispatch ?  
*(1.4.1.2) Yes*
- 

- 1.4.1.3 Dual ?  
*(1.4.1.3) Yes*
- 

- 1.4.2 How do you define an interface?  
*(1.4.2) Use the Type Library Editor to add a new interface. Then add methods and properties as appropriate. An interface declaration in Delphi syntax will then be created automatically. You cannot add interfaces to the \_TLB file yourself, since this is created from the editor.*
- 

- 1.4.3 Is it possible to implement multiple interfaces on a component ?  
*(1.4.3) Yes*
- 

- 1.4.3.1 If Yes, how ?  
*(1.4.3.1) Add a new interface via the editor. Select the CoClass to implement it. Select the "members tab". Right click the window. Select "insert interface" from the pop-up-menu.*
- 

- 1.4.4 Predefined interfaces:  
*(1.4.4.1) It is difficult. Delphi will let you open .TLB files, which will result in a new instance of the Type Library Editor showing the library. Drag-and-drop is somewhat supported between instances of the editor, but usually crashes it. You can add references to existing type libraries (on the "members" tab of the*

library), but for some reason this changes nothing with respect to what you can implement.

---

- 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 difficult. Delphi will let you open .TLB files, which will result in a new instance of the Type Library Editor showing the library. Drag-and-drop is somewhat supported between instances of the editor, but usually crashes it. You can add references to existing type libraries (on the "members" tab of the library), but for some reason this changes nothing with respect to what you can implement.*
- 

- 1.4.4.2 What do you use as foundation for the predefined interface ?  
(1.4.4.3) *Since it will ultimately be part of the project as any other type library defined through the wizards, there is no difference here.*
- 

- 1.4.4.2.1 Type lib ?  
(1.4.4.2.1) *Yes and cheat as above.*
- 

- 1.4.4.2.2 IDL ?  
(1.4.4.2.2) *Possible, but you will need the MIDL compiler and cheat as above.*
- 

- 1.4.4.2.3 Source code ?  
(1.4.4.2.3) *No*
- 

- 1.4.4.2.4 other  
(1.4.4.2.4) *No*
- 

- 1.4.4.3 How do you use a predefined interface?
  - 1.4.5 Interface maintenance  
(1.4.5.1) *Yes. Delphi does not distinguish between published and non-published interfaces.*
-

- 1.4.5.1 Which types of changes in a published interface are allowed ?  
(1.4.5.1) *Yes. Delphi does not distinguish between published and non-published interfaces.*
- 

- 1.4.5.1 Insert new methods and/or parameters ?  
(1.4.5.1) *Yes. Delphi does not distinguish between published and non-published interfaces.*
- 

- 1.4.5.2 Change existing methods and/or parameters ?  
(1.4.5.2) *Yes*
- 

- 1.4.5.3 Delete methods and/or parameters ?  
(1.4.5.3) *Yes*
- 

- 1.4.5.4 What are the consequences of changes on the generated component ?  
(1.4.5.4) *When adding entities, corresponding Pascal declarations will be created. When you remove an entity, e.g. an interface, the CoClass units will not be cleaned up, but the \_TLB file will.*
- 

- 1.4.6 Interface optimizations:  
(1.4.6.1) *Yes*
- 

- Which type of interface optimization are supported:
    - 1.4.6.1 None?  
(1.4.6.1) *Yes*
- 

- 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) *C++ specific feature not supported.*

- 1.4.6.3 Other:  
(1.4.6.3) No
- 

1.5 Reference counting  
(1.5.1) -

---

- 1.5.1 A description of aspects of reference counting  
(1.5.1) -
    - 1.5.1.1 This question is intentionally left blank  
(1.5.1.1) This answer is intentionally left blank.
    - 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) Delphi will implicitly perform `AddRef()`, when assignment is made on an interface variable as for example in

```
var
    MyCalc: ICalc;
begin
    MyCalc := CreateComObject(CLASS_CALC) as ICalc;
    ...
end;
```

*In the example the "as" keyword performs an implicit "QueryInterface" for the ICalc interface. "CreateComObject" is a wrapper function for "CoCreateInstance", but Delphi does not force you to use that wrapper function. The above code could also have been written:*

```
var
    MyCalc: ICalc;
begin
    CoCreateInstance(CLASS_CALC, nil, CLSCTX_INPROC_SERVER,
        ICalc, MyCalc);
    ...
end;
```

end;

- 
- 1.5.2.1 Is AddRef called automatically by a generated component when returning an interface reference ?

*(1.5.2.1) Yes*

- 
- 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) No, lest he uses the API functions.*

- 
- 1.5.2.3 What happens when the reference count drops to 0 ?

*(1.5.2.3) The component is released.*

- 
- 1.5.2.3.1 Is it possible to customize what happens when the reference count drops to 0 ?
  - 1.5.2.4 How does the tools mechanism for implementing reference counting affect flexibility etc. ?

*(1.5.2.4) They don't: you can choose the strategy of your preference.*

---

## 1.6 Threading

- 1.6.1 Which threading models are supported ?

*(1.6.1.2) -*

- 
- 1.6.1.2 Not specified ?

*(1.6.1.2) -*

- 
- 1.6.1.3 Single ?

*(1.6.1.3) Yes*

---

- 1.6.1.4 Appartment ?  
(1.6.1.4) No
- 

- 1.6.1.5 Free ?  
(1.6.1.5) No
- 

- 1.6.1.6 Other:  
(1.6.1.6) No
- 

- 1.6.2 Multi threaded components:  
(1.6.2) Delphi has a native wrapper class for Windows threads, the TThread class. You can use this in any Delphi program, including components.
- 

- 1.6.2.1 Can a component be multi-threaded ?  
(1.6.2.1) Yes
- 

- 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) You can also use the API if you prefer.
- 

## 1.7 Throwing errors

(1.7.1) All COM components in Delphi descend from the TComObject class, which implements ISupportErrorInfo as well as IUnknown.

---

- 1.7.1 Do generated components support:  
(1.7.1) All COM components in Delphi descend from the TComObject class, which implements ISupportErrorInfo as well as IUnknown.
- 

- 1.7.1.1 ISupportErrorInfo ?  
(1.7.1.1) Yes
-

- 1.7.1.2 this question is intentionally left blank  
(1.7.1.2) -
- 

- 1.7.2 Direct support for HRESULT  
(1.7.2) *Delphi wraps HRESULTS into native exceptions.*
- 

- 1.7.2.1 Does the tool support the use of existing HRESULT's ?  
(1.7.2.1) *Yes, the constants are declared in the include file "ActiveX.pas"*
- 

- 1.7.2.2 Does the tool support the use of interface dependent HRESULTS ?  
(1.7.2.2) *You can create your own as needed.*
- 

- 1.7.2.3 Does the tool provide support for HRESULT creation ?  
(1.7.2.3) *No.*
- 

- 1.7.2.4 Is it easy to create/use HRESULTS ?  
(1.7.2.4) *Easy to use, harder to define.*
- 

- 1.7.3 Are there mappings between HRESULTS and native language exceptions ?  
(1.7.3) *Yes, as described above.*
- 

- 1.7.3.1 Is the mapping manual ?  
(1.7.3.1) *Yes*
- 

- 1.7.3.2 Is the mapping automatic ?  
(1.7.3.2) *Yes*
-

- 1.7.4 How does the mapping work ?  
(1.7.4) Components that return errorcodes or exceptions through *IErrorInfo* interfaces, will have their errors translated into native Delphi exceptions. A base class for these are the *EOleException* exception class.

## 1.8 Datatypes (1.8.1)

- 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 following table lists the mapping of simple data-types

<b>COM Constant</b>	<b>Delphi Type</b>	<b>Description</b>
<i>VT_I2</i>	<i>SmallInt</i>	2 byte signed int
<i>VT_I4</i>	<i>Integer</i>	4 byte signed int
<i>VT_R4</i>	<i>Single</i>	4 byte real
<i>VT_R8</i>	<i>Double</i>	8 byte real
<i>VT_CY</i>	<i>Currency</i>	currency
<i>VT_DATE</i>	<i>TDateTime</i>	date
<i>VT_DISPATCH</i>	<i>IDispatch</i>	<i>IDispatch FAR*</i>
<i>VT_ERROR</i>	<i>SCODE</i>	<i>SCODE</i>
<i>VT_BOOL</i>	<i>WordBool</i>	True=-1, False=0
<i>VT_UNKNOWN</i>	<i>IUnknown</i>	<i>IUnknown FAR*</i>
<i>VT_DECIMAL</i>	<i>TDecimal</i>	16 byte fixed point
<i>VT_I1</i>	<i>ShortInt</i>	signed char
<i>VT_UI1</i>	<i>Byte</i>	unsigned char
<i>VT_UI2</i>	<i>Word</i>	unsigned short
<i>VT_UI4</i>	<i>UINT</i>	unsigned long
<i>VT_I8</i>	<i>Comp</i>	signed 64-bit int
<i>VT_UI8</i>	<i>LargeUINT</i>	unsigned 64-bit int
<i>VT_INT</i>	<i>SYSINT</i>	signed machine int
<i>VT_UINT</i>	<i>SYSUINT</i>	unsigned machine int
<i>VT_HRESULT</i>	<i>HResult</i>	<i>HResult</i>

- 1.8.1.2 Strings ?

(1.8.1.2) Strings are mapped as follows:

<b>COM Constant</b>	<b>Delphi Type</b>	<b>Description</b>
<i>VT_BSTR</i>	<i>WideString</i>	Automation string
<i>VT_LPSTR</i>	<i>PChar</i>	Null terminated C-style
<i>VT_LPWSTR</i>	<i>PWideChar</i>	Null terminated widestring C-style

- 1.8.1.3 Predefined types (automation compliant types)?  
*(1.8.1.5) Will be mapped as described in 1.8.1.3. When a type is not simple (e.g. struct), its name will simply be put in as the datatype in the parameter list.*
- 

- 1.8.1.4 User-defined types (e.g. struct, enum)?  
*(1.8.1.4) Only enums can be created, and these are mapped as described above.*
- 

- 1.8.1.5 Object parameters (interface references) ?  
*(1.8.1.5) Will be mapped as described in 1.8.1.3. When a type is not simple (e.g. struct), its name will simply be put in as the datatype in the parameter list.*
- 

- 1.8.1.6 Variants ?  
*(1.8.1.6) Variants are supported by Delphi, and here known as "OleVariant".*
- 

- 1.8.1.7 SafeArrays ?  
*(1.8.1.7) Safearrays will be mapped into the TSafeArray class, which is a full wrapper for safearrays.*
- 

## 1.9 Marshalling

- 1.9.1 How can marshalling be handled:  
*(1.9.1.1) See below.*
- 

- 1.9.1.1 Don't know! ?  
*(1.9.1.1) See below.*
-

- 1.9.1.2 OLE32.DLL ?  
*(1.9.1.2) Yes, if you provide dual interfaces or ones based on the automation types.*
- 

- 1.9.1.3 Own Proxy/Stub DLL ?  
*(1.9.1.3) Yes, but you must use MIDL to get the proxy/stub files and then a C-compiler to build it.*
- 

- 1.9.1.3.1 Can it be attached to the component DLL?  
*(1.9.1.3.1) No, use MIDL as described above.*
- 

- 1.9.1.4 Directly made by tool ?  
*(1.9.1.4) No.*
- 

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

- 1.9.2 Does the tool support IMarshal ?  
*(1.9.2) You can provide your own implementation of IMarshal if you care.*
- 

1.10 How does the server support typelibraries  
*(1.10.1) No*

---

- 1.10.1 IDL Integration ?  
*(1.10.1) No*
- 

- 1.10.1.1 IDL/ODL ?  
*(1.10.1.1) No.*
-

- 1.10.2 Implicit creation of typelibraries ?  
*(1.10.2) Through the Type Library Editor.*
- 
- 1.10.3 Integration of typelibraries as resource in server ?  
*(1.10.3) Yes*
- 
- 1.10.4 Are errors in the type library able to "cheat" the tools ?  
*(1.10.4) No, the compiler will (and does) complain.*
- 

## 2. Compilation and Distribution

2.1 Is version compatibility checking performed during compilation and how ?  
*(2.1) No*

---

2.2 Interface compatibility  
*(2.2.1) Yes*

---

- 2.2.1 Is it possible to change an interface and keep the same IID?  
*(2.2.1) Yes*
- 

2.3 Which types of target code does the tool produce (i.e. native/optimized native + runtime, p-code, nothing/interpreted) ?  
*(2.3) You can build COM servers that will need the Delphi runtime libraries. These servers are very small, compared to ones that link the runtime libraries into them. The optimisations that Delphi supports for all programs does also apply to components.*

---

2.4 How does the tool support registration of components ?  
*(2.4.1) Delphi components that are out-of-process will register themselves automatically when executed as if they were ordinary programs. In-process servers can be registered through REGSVR32.EXE.*

---

- 2.4.1 Components are selfregistering ?  
*(2.4.1) Delphi components that are out-of-process will register themselves automatically when executed as if they were ordinary programs. In-process servers can be registered through REGSVR32.EXE.*
- 

- If Yes then
    - 2.4.1.1 DLL RegSvr32 ?  
*(2.4.1.1) Yes*
- 

- 2.4.1.2 EXE: selfregistering when run once ?  
*(2.4.1.2) Yes*
- 

- 2.4.1.3 EXE: selfregistering when compiled ?  
*(2.4.1.3) No*
- 

- 2.4.2 Registration is done through  
*(2.4.2.1) No*
- 

- 2.4.2.1 Installation ?  
*(2.4.2.1) No*
- 

- 2.4.2.2 Has to be done manually ?  
*(2.4.2.2) Yes*
- 

- 2.4.3 Does the tool support unregistration of components ?  
*(2.4.3) Yes. In-process servers may be unregistered either from the menus in Delphi or via REGSVR32.EXE, using the /u option. Out-of-process servers can be registered by running them once and unregistered again by supplying the switch "/unregserver" on the commandline.*
- 

- 2.4.4 Does the tool support registration using ProgID ?  
*Yes.*

- 
- 2.4.4.1 Can the developer freely choose a components ProgID ?  
*The ProgID will be chosen by the IDE on basis of the project and component names.*
  - 2.4.5 Does the tool support version independent ProgID's, and if yes how ?  
*No.*

2.5 Support for component categories:

*(2.5.1) Delphi has no direct support for grouping components into categories.*

---

- 2.5.1 Is it possible to define and implement your own component categories ?  
*(2.5.1) Delphi has no direct support for grouping components into categories.*
- 

- 2.5.1.1 How ?  
*(2.5.1.1) -*
- 

- 2.5.2 Does the tool have support for predefined component categories and how ?  
*(2.5.2) No.*
- 

2.6 Can the developer freely choose CLSID for a component ?  
*(2.6) Yes.*

---

## **3. Component Client**

### **3.1 Support for COM clients**

#### 3.1.1 Creating and deleting components

*(3.1.1.1) Yes: use the wizards.*

---

- 3.1.1.1 Is it easy to create new components ?  
*(3.1.1.1) Yes: use the wizards.*
-

- 3.1.1.2 Is it easy to delete existing components ?  
(3.1.1.2) *Yes: use the Type Library Editor.*
- 

- 3.1.1.3 Does the tool delete registry entries for deleted components after recompilation ?  
(3.1.1.3) *Yes, but not until the component has been re-registered.*
- 

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

### 3.1.2 COM Library Support

(3.1.2.1) *Yes, but you will not get the benefits of automatic reference counting if you do so.*

---

- 3.1.2.1 Does the tool allow direct access to COM functionality, like calling COM functions direct (CoCreateInstance, CreateInstance, ...). ?  
(3.1.2.1) *Yes, but you will not get the benefits of automatic reference counting if you do so.*
- 

- 3.1.2.2 Does the tool encapsulate COM functionality (indirect COM support) ?  
(3.1.2.2) *Yes, as described in 1.5.2.*
- 

- 3.1.2.3 Does the tool provide COM Helper Classes ?  
(3.1.2.3) *Yes. There are a number of classes, encapsulating simple COM servers, automation servers, and even Class Factories.*
- 

- 3.1.2.4 Is it possible to combine native and encapsulated COM support ?  
(3.1.2.4) *Yes.*
- 

### 3.1.3 Acquisition of interfaces

(3.1.3.1) *Yes.*

---

- 3.1.3.1 Is it possible to call QueryInterface directly ?  
(3.1.3.1) Yes.
- 

- 3.1.3.2 Does the tool encapsulate QueryInterface ?  
(3.1.3.2) Yes, using the "as" operator.
- 

- 3.1.3.3 Reference counting ?  
(3.1.3.3) Implicit through assignment or explicit via AddRef and Release.
- 

- 3.1.3.3.1 Is AddRef called automatically by a generated client when using an interface as parameter for a (server) method ?
- 3.1.3.3.2 Is Release called automatically by a generated client when exiting scope for an interface reference ?

#### 3.1.4 Component binding

(3.1.4.1) Yes.

---

- 3.1.4.1 Is early binding supported through: ?  
(3.1.4.1) Yes.
- 

- 3.1.4.1.1 VTable binding ?  
(3.1.4.1.1) Yes.
- 

- 3.1.4.1.2 DispID binding ?  
(3.1.4.1.2) Yes.
- 

- 3.1.4.2 Is late binding supported and how ?  
(3.1.4.2) Yes.
- 

- 3.1.4.3 Binding through ProgID  
(3.1.4.3.1) Yes. You can use both GUIDs and ProgIDs at your leisure

- 3.1.4.3.1 Does the tool support binding of (server) component through version independent ProgID ?  
*(3.1.4.3.1) Yes. You can use both GUIDs and ProgIDs at your leisure*
- 3.1.4.3.2 Does the tool support binding of (server) component through version specific ProgID ?  
*(3.1.4.3.2) Yes.*

3.1.5 Is the process of catching errors supported through:

*(3.1.5) Error handling is supported through Delphi exceptions. The mapping from HRESULTs or IErrorInfo is implicit, but can also be used explicitly.*

---

- 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) All components will have native support for being the inner of an aggregate. If you want to create a component that aggregates another one, you will have to override the default implementation of QueryInterface.*

---

- 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) No.*
- 

- 3.2.1.3 Is Manual programming of aggregation Easy/Difficult ?  
*(3.2.1.3) The implementation of QueryInterface can be defined as shown below. You will also need to create the aggregates in the initialization procedure. This procedure is automatically called by the Class Factory, when a new instance of a Delphi COM class is made.*

```
procedure TClass2.Initialize;
begin
  { Initialize aggregated components }
  punkOuter := Self;
  OleCheck(CoCreateInstance(Class_ItemInfo, punkOuter,
    CLSCTX_INPROC_SERVER, IUnknown, punkInnerItemInfo));
end;

function TClass2.QueryInterface(const IID: TGUID; out
Obj): Integer;
begin
  {Is interface implemented by this class?}
  if GetInterface(IID, Obj) then
    Result := S_OK
  {Is interface implemented by aggregated class?}
  else if (pUnkInnerItemInfo.QueryInterface(IID, Obj) =
S_OK) then
    Result := S_OK
  else
    Result := E_NOINTERFACE;
end;
```

---

### 3.3 Delegation

3.3.1 Support for delegation is:

*(3.3.1.1) Yes. It is quite simple to add delegates to a COM class: Simply create the instances in the Initialize procedure and forward the calls in the method implementations as shown below. TClass3 contains CoClass Calc, and delegates calls on the ICalc interface as shown in the implementation of Modulus.*

```
procedure TClass3.Initialize;
begin
  { Launch contained components }
  Class3ICalc := CreateComObject(Class_Calc) as ICalc;
end;

function TClass3.Modulus(v1: Integer; v2: Integer): Integer; safecall;
begin
  Result := Class3ICalc.Modulus(v1, v2);
end;
```

---

- 3.3.1.1 Manual ?

*(3.3.1.1) Yes. It is quite simple to add delegates to a COM class: Simply create the instances in the Initialize procedure and forward the calls in the method implementations as shown below. TClass3 contains CoClass Calc, and delegates calls on the ICalc interface as shown in the implementation of Modulus.*

```
procedure TClass3.Initialize;  
begin  
    { Launch contained components }  
    Class3ICalc := CreateComObject(Class_Calc) as ICalc;  
end;  
  
function TClass3.Modulus(v1: Integer; v2: Integer): Integer;  
safecall;  
begin  
    Result := Class3ICalc.Modulus(v1, v2);  
end;
```

---

- 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  
(4.1.1) *The Type Library Editor is useful for browsing type libraries, and will let you import these from the Windows Registry or from a file.*

---

- 4.1.1 What kind of object browser facilities are provided ?  
(4.1.1) *The Type Library Editor is useful for browsing type libraries, and will let you import these from the Windows Registry or from a file.*
- 

- 4.1.2 What kind of syntax help/info for COM-components and methods herein ?  
(4.1.2) *There is no way to use a help-file attached to a component from Delphi, but help-strings will be added as comments in auto-generated code.*
- 

- 4.1.3 Does the tool provide context sensitive help for components (to be used by component integraters) ?  
(4.1.3) *You can specify Helpfiles and context, but not test the functionality.*
- 

- 4.1.4 Support for interface prototypes (components implementing interfaces specified externally)

*(4.1.4) There is no direct IDE support for this, but you can implement any interface in Delphi*

---

- 4.1.4.1 How can you make this manually ?  
*(4.1.4.1) -*
- 

- 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 TypeLibrary Editor can load any TLB file and generate skeleton code for you.*
- 

- 4.1.5 Debugging  
*(4.1.5) For in-process servers, you can specify an application that will be used to debug the component. You can then debug as you would ordinary Delphi applications, using breakpoints and watches in your component. To debug an out-of-process server, it must be launched before any clients of it. If you launch it from Delphi, you can then set breakpoints and watches as usual.*
- 

- 4.1.5.1 Which facilities exist for debugging COM-components ?  
*(4.1.5.1) The ordinary Delphi debugger.*
- 

- 4.1.5.2 Is it possible in the development to debug from client into component ?  
*(4.1.5.2) Yes. You can have multiple instances of Delphi running, and can connect them as described above.*
- 

- 4.1.5.3 If any, what are the differences between debugging in-process and out-of-process servers ?  
*(4.1.5.3) See 4.1.5*
- 

4.2 Evaluation of the development tool  
*(4.2.1) -*

---

- 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) *Yes. All Delphi applications are made using wizards, and so are COM components. When you have defined the type library, the rest is ordinary Delphi programming.*
- 

- 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 help files of Delphi in general lack examples, but by being a little stubborn, it is not so hard.*
- 

- 4.2.1.1.2 Is the COM-based development in the tool similar to non-COM-based development ?  
(4.2.1.1.2) *Yes. Apart from the Type Library Editor, which is easy to use.*
- 

- 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) *When you have figured out to use the wizards, and have prior knowledge of the COM entities (interface, CoClass, etc.), it is easy to use Delphi. An expert may override Borland's standard implementations of interfaces and provide his own. All COM support is wrapped up in Delphi units that are shipped with the tool in both compiled and source-code form. It is therefore a question of reading the source and patching it appropriately.*
- 

## 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.*