OLE 자동화 서버의 작성

(Creating OLE Automation Servers)

자동화 서버는 자동화 컨트롤러라고 불리는 클라이언트 어플리케이션에 메소드를 노출시키는 어플리케이션을 말한다. 컨트롤러는 델파이 뿐만 아니라, 비주얼 베이직이나 C++과 같은 다른 도구로 작성할 수도 있다.
이미 26 장에서 COM 객체 서버를 작성하고 이를 이용하는 방법을 소개한 바 있지만, 여기서는 IDispatch 인터페이스에 기초를 둔 자동화 서버를 이용하게 된다. 자동화 서버는 프로세스간 통신의 방법으로도 최근에 가장 흔히 사용하는 것이므로 여기에 대해서는 잘 알아두는 것이 좋을 것이다.

자동화 객체의 제작

오브젝트 파스칼에서 자동화 객체는 TAutoObject 클래스에서 상속받아서 작성하게 된다. TAutoObject 객체는 자동화 프로토콜을 지원하며, 이를 이용하여 어플리케이션에서 사용할 수 있다. 자동화 객체는 자동화 객체 위저드를 이용해서 쉽게 만들 수 있다.
자동화 객체를 생성하기 전에, 먼저 사용할 프로젝트 객체를 정해야 한다. 여기에서는 File|New 메뉴를 선택하고, ActiveX 탭에서 ActiveX Library 를 이용하도록 한다. 물론 프로젝트는 이런 액티브 X DLL 라이브러리가 아니어도 상관이 없다.
자동화 객체 위저드를 이용하는 방법은 File|New 메뉴를 선택하고, ActiveX 탭에서 Automation Object 를 선택한다. ComObject 위저드와 마찬가지로 Automation Object 위저드에서도 클래스의 이름과 인스턴싱 type, 쓰레딩 모델을 선택해야 한다. 여기에 대해서는 26 장의 내용을 참고하기 바란다.
그 밖에 Generate event support code 옵션을 선택할 수 있는데, 이 옵션을 선택하면 위저드가 자동하 객체의 이벤트를 처리하기 위해 분리된 인터페이스를 구현한다는 뜻이다.
이들 항목을 모두 선택하고 OK 버튼을 클릭하면 현재 프로젝트에 자동화 객체가 추가되면서 타입 라이브러리가 열리는 것을 볼 수 있을 것이다. 타입 라이브러리에서 프로퍼티와 인터페이스의 메소드를 설정하면 된다.
기본적으로 델파이의 자동화 객체는 듀얼 인터페이스를 구현한다. 그러므로, IDispatch 인터페이스를 통한 late(런타임) 바인딩과 VTable 을 통한 early(컴파일-타임) 바인딩을 모두 지원하게 할 수 있다.

어플리케이션 프로퍼티, 메소드, 이벤트의 설정

타입 라이브러리에 담겨진 정보는 호스트 어플리케이션이 주어진 객체가 어떤 기능을 할 수 있는지를 알 수 있도록 해준다. 자동화 서버를 작성할 때에는 타입 라이브러리 정보가 어플리케이션에 리소스로서 자동으로 컴파일 된다.

- 프로퍼티 노출

프로퍼티는 객체의 색상이나 폰트와 같이 객체의 상태에 대한 정보를 설정하거나, 반환하는 멤버이다. 예를 들어, 버튼 컨트롤에는 다음과 같은 프로퍼티가 선언되어 있다.

```
property Caption: WideString;
```

자동화에 대한 프로퍼티를 노출시키려면 타입 라이브러리 에디터에서 자동화 객체에 대한 인터페이스를 선택하고, 툴바의 Property 버튼을 클릭하거나 이 버튼의 옆의 화살표를 클릭하고 노출한 프로퍼티의 type 을 클릭한다. 그리고, Attributes pane 에서 프로퍼티의 이름을 지정하고, Parameters pane 에서 프로퍼티의 리턴 type 을 지정하고 적당한 파라미터를 추가하면 된다. 그리고, Refresh 버튼을 클릭한다.
이렇게 하면, 프로퍼티에 대한 정의와 구현부의 뼈대 코드가 자동화 객체의 유닛 파일에 추가될 것이다. 마지막으로 만들어진 뼈대 코드에 프로퍼티의 구현을 위한 코드를 추가하면 된다.

- 메소드 선언

메소드는 프로시저나 함수일 수 있다. 메소드를 노출시키려면 타입 라이브러리 에디터에서 인터페이스를 선택하고 Method 버튼을 클릭한다. 그리고, Attributes pane 에서 메소드의 이름을 지정하고 Parameters pane 에서 메소드의 리턴 type 과 적절한 파라미터를 추가한 후 Refresh 버튼을 클릭한다.
이렇게 하면, 프로퍼티와 마찬가지로 자동화 객체 유닛 파일에 메소드에 대한 선언부와 구현부의 뼈대 코드가 추가된다. 마지막으로 여기에 실제 기능을 구현할 코드를 삽입하면 된다.

- 이벤트 노출

자동화 객체에 이벤트를 노출시키려면 처음에 자동화 객체 위저드에서 Generate event support code 옵션을 체크해야 한다. 이렇게 하면, Events 인터페이스를 포함한 자동화 객체를 생성한다.
그리고, 타입 라이브러리 에디터에서 자동화 객체에 대한 Events 인터페이스를 선택한다.

여기에서 Method 버튼을 클릭하고 Attributes pane 에서 이벤트의 적절한 이름을 지정한다. 그리고, Refresh 버튼을 클릭하면 이벤트에 대한 선언부와 뼈대 코드가 자동화 객체 유닛 파일에 삽입될 것이다.
코드 에디터에서 TAutoObject 에서 상속받은 이벤트 핸들러를 다음과 같은 형식으로 추가한다.

```
unit ev;

interface
uses
  ComObj, AxCtrls, ActiveX, Project1_TLB;

type
  TMyAutoObject = class (TAutoObject, IConnectionPointContainer, IMyAutoObject)
private
  ...
public
  prodcure Initialize; override;
  procedure EventHandler;                                    {이벤트 핸들러}
```

그리고, Initialize 메소드의 후반부에 다음과 같이 이벤트를 이벤트 핸들러에 연결한다.

```
procedure TMyAutoObject.Initialize;
begin
  inherited Initialize;
  FConnectionPoints:= TConnectionPoints.Create(Self);
  if AutoFactory.EventTypeInfo <> nil then
    FConnectionPoints.CreateConnectionPoint (AUtoFactory.EventIID,
      ckSingle, EventConnect);
  OnEvent = EventHandler;                                    {이벤트 핸들러와 이벤트를 연결한다}
end;
```

VCL 객체를 이용하는 자동화에 대한 이벤트를 노출시키는 것은, 다음과 같이 이벤트를 변경하여 설정한다.

```
procedure TMyAutoObject.EventHandler;
```

```
begin
  if FEvents <> nil then FEvents.MyEvent;                    {이벤트 핸들러를 노출시킨다}
end;
```

자동화 객체에서 이벤트 처리하기

델파이 4 에서의 자동화 객체 위저드에서 Generates event code 옵션을 체크하면 기본적인 이벤트 코드를 생성해준다.
이벤트를 처리하기 위해서는 처리할 이벤트의 종류에 따라 다른 인터페이스와 함께 액티브X IConnectionPoint 와 IConnectionPointContainer 인터페이스를 지원해야 한다.
IConnectionPoint 인터페이스는 객체에 대한 outgoing 포인터를 노출하도록 허용하는 역할을 한다. 이러한 포인터를 이벤트 싱크(event sink)라고 한다. IConnectionPointContainer 인터페이스는 객체에 의해 제공되는 커넥션 포인트(connection point)를 나열하고, 호출자가 여기에서 적절한 커넥션 포인트를 찾을 수 있도록 해준다. 이벤트가 발생하면, 커넥션을 이용해서 이벤트를 처리하는 인터페이스를 호출하게 되는 이벤트 소스와 인터페이스를 실제 구현하는 이벤트 싱크를 연결하는 것이다. 싱크는 이벤트의 세트에 대한 멤버 함수들을 구현한다. 여기에 대해서는 30 장에서 더욱 자세히 다루게 될 것이다.

자동화 인터페이스

델파이 위저드는 듀얼 인터페이스를 디폴트로 구현한다. 즉, 자동화 객체가 IDispatch 인터페이스를 통해 런타임에서 late 바인딩을 지원하며, 직접 객체의 VTable 의 멤버 함수를 호출할 수 있도록 하여 컴파일 시의 early 바인딩을 지원한다.

- 듀얼 인터페이스 (Dual interface)

듀얼 인터페이스는 동시에 사용자 정의 인터페이스와 dispinterface 를 지원한다.
VTable 인터페이스를 위해서 컴파일러는 형 검사를 실시하고, 보다 정확한 에러 메시지를 제공하며, 데이터 형 정보를 얻을 수 없는 자동화 컨트롤러를 위해 dispinterface 가 객체에 런타임에서 접근할 수 있도록 허용한다.
듀얼 인터페이스 서버를 in-process 서버로 구현할 경우에는 vtable 인터페이스를 통해 보다 빠르게 접근할 수 있도록 하며, out-of-process 서버의 경우에는 COM 이 vtable 인터페이스와 dispinterface 에 대한 데이터를 타입 라이브러리에서 제공하는 정보에 따라 마샬링 해준다.
듀얼 인터페이스 vtable 의 처음 3 개의 엔트리는 IUnknown 인터페이스의 것이며, 그 다음 4 개의 엔트리는 IDispatch 인터페이스를 지원한다. 나머지 내용이 사용자 정의 인터페이

스의 메소드에 직접 전근할 때 사용되는 COM 엔트리이다.

- 디스패치 인터페이스 (Dispatch interface)

자동화 컨트롤러는 COM 의 IDispatch 인터페이스를 이용하여 COM 서버 객체에 접근한다. 컨트롤러는 먼저 객체를 생성하고, 객체의 IUnknown 인터페이스를 이용하여 IDispatch 인터페이스에 대한 포인터를 질의한다. IDispatch 인터페이스는 내부적으로는 디스패치 ID(dispID)를 사용하여 메소드나 프로퍼티를 추적한다. dispID 는 인터페이스 멤버에 대한 유일한 식별자로 쓰인다. IDispatch 를 이용하여 컨트롤러는 객체에 대한 데이터 형 정보를 얻게 되고, 이를 지정된 dispID 에 매핑하여 사용하게 된다. dispID 는 런타임에서 IDispatch 인터페이스의 GetIDsOfNames 메소드를 이용하여 얻을 수 있다.
일단 dispID 를 얻게 되면, 컨트롤러는 IDispatch 인터페이스의 Invoke 메소드를 이용하여 적절한 코드를 실행하게 된다. 이 과정에서 프로퍼티나 메소드의 파라미터 들을 Invoke 메소드의 파라미터로 패키징한다. Invoke 메소드는 고정된 컴파일 타임 signature 를 가지고 있으며 호출하는 인터페이스 메소드의 파라미터의 수와 내용이 어떤 것이든 받아들인다.

- 사용자 정의 인터페이스 (Custom interface)

사용자 정의 인터페이스는 클라이언트가 vtable 의 나열된 순서에 따라서 메소드를 호출하게 되며, 파라미터에 대한 데이터 형에 대한 정보를 호출자가 알고 있다는 기본 과정에서 출발하게 된다. VTable 은 객체의 프로퍼티와 메소드에 대한 주소를 모두 나열하게 되는데, 객체가 IDispatch 인터페이스를 지원하지 않을 경우에는 객체의 IUnknown 인터페이스를 따른다.
클라이언트가 객체에 대한 타입 라이브러리를 얻을 수 있으면, IDispatch 인터페이스의 dispID 를 가지고 직접 vtable 의 오프셋(offset)에 직접 바인딩하게 된다. 이런 컴파일 타임 바인딩은 객체의 메소드나 프로퍼티를 직접 객체의 vtable 로 호출하게 된다. 그러므로, Invoke 나 GetIDsOfNames 메소드를 호출할 필요가 없다.

데이터 마샬링 (Marshaling data)

out-of-process 또는 원격 서버인 경우에는 COM 이 어떤 방식으로 데이터를 마샬링하는지 이해하고 있어야 한다. IDispatch 인터페이스를 사용하거나, 서버를 제작할 때 타입 라이브러리를 작성하고 인터페이스를 OLE 자동화 플래그로 설정하면 이를 자동으로 지원할 수 있다. COM 은 자동화 호환 데이터 형에 대해서는 타입 라이브러리를 이용할 경우 프록시와 스텁을 설정하여 데이터를 마샬링할 수 있다.
마샬링을 직접 지원하고자 하면 IMarshal 인터페이스의 모든 메소드를 구현해야 하는데, 이

를 커스텀 마샬링이라고 한다.

- 자동화 호환 데이터 형 (Automation compatible types)

함수의 결과와 파라미터의 데이터 형이 듀얼과 dispatch 인터페이스인 경우에는 반드시 자동화 호환 데이터 형이어야 한다. OLE 자동화 호환 데이터 형으로는 다음과 같은 것들이 있다.

1. SmallInt, Integer, Single, Double, WideString 과 같은 단순 데이터 형. 자세한 것은 도움말을 참고하기 바란다.
2. 타입 라이브러리에 정의된 열거형(Enumeration type). OLE 자동화 호환 열거형은 32 비트 값으로 저장되며, 정수형으로 취급된다.
3. OLE 자동화에 문제가 없는 IDispatch 에서 상속받은 인터페이스 데이터 형
4. 타입 라이브러리에 정의된 Dispinterface 데이터 형
5. IFont, IStrings, IPicture 등은 각각 TFont, TStrings, TPicture 와 같은 helper 객체와 매핑된다.

액티브 X 컨트롤과 액티브폼 위저드는 필요할 때 이런 helper 객체를 생성한다. 이런 helper 객체를 사용하려면 GetOleFont, GetOleStrings, GetOlePicture 와 같은 전역 루틴을 사용한다.

- 자동 마샬링에 대한 데이터 형 제한

인터페이스의 마샬링을 자동으로 지원하게 하기 위해서는 다음과 같은 제한 사항을 지켜야 한다. 자동화 객체를 타입 라이브러리 에디터를 이용해서 편집할 때에는 이런 제한점을 지키도록 도와준다.

1. 데이터 형은 플랫폼간 통신(cross-platform communication)에 호환되어야 한다.
2. 듀얼 인터페이스의 모든 멤버는 반드시 함수의 리턴값으로 HRESULT 를 넘겨야 한다.
3. 다른 값을 반환해야 하는 듀얼 인터페이스의 멤버는 이를 var 또는 out 파라미터로 지정해서 사용해야 한다.

- 커스텀 마샬링 (Custom marshaling)

보통의 경우에는 앞에서 설명한 제한 사항을 지키면서 COM 에서 지원하는 마샬링을 자동으로 사용하게 된다. 어쨌든, 마샬링의 수행 성능을 향상시키기 위해 커스텀 마샬링을 사

용하게 될 수도 있다.

자동화 서버의 등록

자동화 서버는 in-process 서버일 수도 있고, out-of-process 서버일 수도 있다.
in-process 서버(DLL, OCX)를 등록하려면, Run|Register ActiveX Server 메뉴를 선택하면 되며 이를 해제하려면 Run|Unregister ActiveX Server 메뉴를 이용하면 된다.
out-of-process 서버를 등록하려면 서버를 실행할 때 /regserver 커맨드 라인 옵션을 사용하면 된다. 커맨드 라인 옵션을 설정할 때에는 Run|Parameters.. 메뉴의 대화 상자를 이용한다. 또한 이를 해제할 때에는 /unregserver 커맨드 라인 옵션을 사용하면 된다.

어플리케이션의 테스트와 디버깅

자동화 서버의 테스트와 디버깅을 위해서는 먼저 Project|Options 대화 상자의 Compiler 탭을 이용하여 디버깅 정보를 사용하도록 설정하고, Tools|Debugger Options 대화 상자에서 Integrated Debugging 을 이용하도록 하면 된다.
모든 in-process 서버는 디버깅과 테스트를 위해 Run|Parameters 메뉴를 선택하고 여기에서 Host Application 박스에 자동화 컨트롤러의 이름을 입력하고 OK 를 클릭한 뒤에 이를 실행하면 된다. 디버깅 방법은 일반적인 어플리케이션과 마찬가지로 정지점(break point)를 이용하는 것이 보통이다.

타입 라이브러리 에디터의 활용

델파이 4 에서 제공되는 타입 라이브러리 에디터는 이전 버전의 타입 라이브러리 에디터의 한계점을 많이 극복한 도구로, COM/CORBA 개발에 있어서 델파이가 가지고 있는 최대의 강점이라고 할 수 있다. 타입 라이브러리는 액티브 X 컨트롤이나 서버에 노출되는 객체 클래스, 멤버 함수, 인터페이스와 데이터 형에 대한 정보를 포함하고 있는 파일을 말한다.
타입 라이브러리는 IDL(Interface Definition Language)나 ODL(Object Description Language)을 이용하여 스크립트를 작성한 후, 이를 컴파일하는 것이 전통적인 개발 방식이다. 그렇지만, 델파이 4 가 제공하는 타입 라이브러리 에디터를 이용하면 쉽게 타입 라이브러리를 작성할 수 있다.
COM 객체나 액티브 X 컨트롤, 자동화 객체를 델파이의 위저드를 이용하여 제작했으면, 타입 라이브러리 에디터는 기본적으로 파스칼 문법으로 된 타입 라이브러리의 유닛 파일을 생성해준다. 타입 라이브러리 에디터에서 인터페이스나 메소드, 프로퍼티 등을 추가하거나 업데이트하고 Refresh 버튼을 클릭하면 이런 변화가 파스칼 유닛 파일에 그대로 반영한다.

● 오브젝트 파스칼과 IDL 문법

디폴트로 타입 라이브러리 에디터의 Text 페이지에는 타입 정보에 대한 내용을 오브젝트 파스칼 문법이나 IDL 문법으로 볼 수 있다. 어떤 언어를 디폴트로 할 것인지를 결정하려면 Tools|Environment Options 메뉴의 Type Library 페이지에서 선택할 수 있다.
오브젝트 파스칼 어플리케이션과 마찬가지로 타입 라이브러리의 식별자(identifier) 역시 대소문자를 가리지 않으며, 255 자까지 사용할 수 있다.
또한, 오브젝트 파스칼 문법으로 정의된 타입 라이브러리 정보는 타입 라이브러리 에디터의 가장 우측의 툴 버튼인 Export to IDL 버튼을 클릭하면 쉽게 IDL 문법으로 전환할 수 있다. 이때 우측의 화살표 키를 클릭하면 COM 을 지원하는 MIDL 문법을 따르게 할 것인지, CORBA 를 지원하는 CORBA IDL 문법을 따르게 할 것인지를 결정할 수 있다.

1. 속성 스펙 (Attribute specifications)

오브젝트 파스칼은 타입 라이브러리에 속성 스펙을 포함하도록 확장되었다. 속성 스펙은 쉼표로 분리된 대괄호로 표현되며, 각각의 속성 스펙은 속성 이름과 값으로 구성된다. 다음에 속성의 이름과 사용 예에 대해서 나열하였다.

속성 이름	사용 예	적용 대상
aggregatable	[aggregatable]	typeinfo
appobject	[appobject]	CoClass typeinfo
bindable	[bindable]	CoClass 멤버를 제외한 멤버
control	[control]	type library, typeinfo
custom	[custom '{7B5687A1-F4E9-11D1-92A8-00C04F8C8FC4}' 0]	anything
default	[default]	CoClass 멤버
defaultbind	[defaultbind]	CoClass 멤버를 제외한 멤버
defaultcollection	[defaultcollection]	CoClass 멤버를 제외한 멤버
defaultvtbl	[defaultvtbl]	CoClass 멤버
dispid	[dispid]	CoClass 멤버를 제외한 멤버
displaybind	[displaybind]	CoClass 멤버를 제외한 멤버
dllname	[dllname 'Helper.dll']	module typeinfo
dual	[dual]	interface typeinfo
helpfile	[helpfile 'c:₩help₩myhelp.hlp']	type library
helpstringdll	[helpstringdll 'c:₩help₩myhelp.dll']	type library

helpcontext	[helpcontext 2005]	CoClass 멤버, 파라미터 제외
helpstring	[helpstring 'payroll interface']	CoClass 멤버, 파라미터 제외
helpstringcontex	[helpstringcontext $17]	CoClass 멤버, 파라미터 제외
hidden	[hidden]	파라미터 제외
immediatebind	[immediatebind]	CoClass 멤버를 제외한 멤버
lcid	[lcid $324]	type library
licensed	[licensed]	type library, CoClass typeinfo
nonbrowsable	[nonbrowsable]	CoClass 멤버를 제외한 멤버
nonextensible	[nonextensible]	interface typeinfo
oleautomation	[oleautomation]	interface typeinfo
predeclid	[predeclid]	typeinfo
propget	[propget]	CoClass 멤버를 제외한 멤버
propput	[propput]	CoClass 멤버를 제외한 멤버
propputref	[propputref]	CoClass 멤버를 제외한 멤버
public	[public]	alias typeinfo
readonly	[readonly]	CoClass 멤버를 제외한 멤버
replaceable	[replaceable]	CoClass 멤버, 파라미터 제외
requestedit	[requestedit]	CoClass 멤버를 제외한 멤버
restricted	[restricted]	파라미터 제외
source	[source]	모든 멤버
uidefault	[uidefault]	CoClass 멤버를 제외한 멤버
usesgetlasterror	[usesgetlasterror]	CoClass 멤버를 제외한 멤버
uuid	[uuid '{7B5687A1-F4E9-11D1-92A8-00C04F8C8FC4}']	type library, typeinfo (required)
vararg	[vararg]	CoClass 멤버를 제외한 멤버
version	[version 1.1]	type library, typeinfo

2. 인터페이스 문법

인터페이스 데이터 형 정보를 오브젝트 파스칼 문법으로 표현하면 다음과 같다.

```
interfacename = interface[(baseinterface)] [attributes]
  functionlist
  [propertymethodlist]
end;
```

예를 들어, 다음의 텍스트는 2 개의 메소드와 1 개의 프로퍼티를 가지고 있는 인터페이스의 선언부이다.

```
Interface1 = interface (IDispatch)
  [uuid '{7B5687A1-F4E9-11D1-92A8-00C04F8C8FC4}', version 1.0]
  function Calculate(optional seed: Integer = 0): Integer;
  procedure Reset;
  procedure PutRange(Range: Integer) [propput, dispid $00000005]; stdcall;
  function GetRange: Integer [propget, dispid $00000005]; stdcall;
end;
```

이를 적절한 IDL 문법으로 변경하면 다음과 같다.

```
[uuid (5FD36EEF-70E5-11D1-AA62-00C04FB16F42), version 1.0]
interface Interface1: IDispatch
{
  longCalculate([in, optional, defaultvalue(0)] long seed);
  void Reset(void);
  [propput, id(0x00000005)] void _stdcallPutRange([in] long Value);
  [propput, id(0x00000005)] void _stdcall getRange([out, retval] long *Value);
 };
```

3. 디스패치 인터페이스 문법 (Dispatch interface syntax)

dispinterface 를 선언하기 위한 오브젝트 파스칼의 문법은 다음과 같다.

```
dispinterfacename = dispinterface [attributes]
  functionlist
  [propertylist]
end;
```

예를 들어, 다음의 텍스트는 앞에서 선언한 인터페이스에 대한 디스패치 인터페이스를 표현한 것이다.

```
MyDispObj = dispinterface
```

```
  [uuid '{5FD36EEF-70E5-11D1-AA62-00C04FB16F42}',
   version 1.0,
   helpstring 'MyObj 에 대한 디스패치 인터페이스']
  function Calculate(seed: Integer): Integer [dispid 1];
  procedure Reset [dispid 2];
  property Range: Integer [dispid 3];
end;
```

여기에 해당하는 IDL 문법은 다음과 같다.

```
[uuid (5FD36EEF-70E5-11D1-AA62-00C04FB16F42),
 version 1.0,
 helpstring("MyObj 에 대한 디스패치 인터페이스")]
 dispinterface Interface1
{
  methods:
  [id(1)] intCalculate([in] int seed);
  [id(2)] void Reset(void);
  properties:
  [id(3)] int Value;
 };
```

4. CoClass 문법 (CoClass syntax)

CoClass 의 타입 정보에 대한 오브젝트 파스칼의 문법은 다음과 같은 형태를 가진다.

```
classname = coclass(interfacename[interfaceattributes], ...); [attributes];
```

예를 들어, 다음의 텍스트는 IMyInt 라는 인터페이스와 DmyInt 라는 dispinterface 에 대한 CoClass 를 선언한 것이다.

```
myapp = coclass(IMyInt [source], DMyInt);
[uuid '{2MD36ABF-90E3-11D1-AA75-02C04FB73F42}',
 version 1.0,
 helpstring 'CoClass',
 appobject]
```

여기에 해당되는 IDL 문법은 다음과 같다.

```
[uuid (2MD36ABF-90E3-11D1-AA75-02C04FB73F42),
 version 1.0,
 helpstring (“CoClass”),
 appobject]
 coclass myapp
{
   methods:
   [source] interfaceIMyInt);
   dispinterface DMyInt;
};
```

4. 그 밖에 ...

그 밖에도 열거형, 앨리어스, 레코드, 모듈 등의 여러가지 데이터 형에 대한 오브젝트 파스칼의 타입 라이브러리 정보에 대한 문법이 존재한다. 여기서 이들 모두에 대해서 다루는 것은 다소 지면 낭비라 생각되므로, 이 정도로 정리하도록 하겠다.
IDL 문법에 대한 내용은 COM/CORBA 에 대해 자세히 접근한 서적에서 참고하기 바라며, 오브젝트 파스칼 문법 부분은 델파이의 도움말에서 제공되는 내용을 참고하면 될 것이다.

- 타입 라이브러리 에디터의 사용법

그러면, 간단한 타입 라이브러리 에디터의 사용방법을 익혀보도록 하자. 앞서 설명한 여러가지 문법 들은 실제로 작성할 필요가 없이, 타입 라이브러리 에디터를 이용하면 자동으로 생성되는 내용들이다.
타입 라이브러리에 인터페이스에 대한 정보를 추가하려면, 툴바에서 인터페이스 아이콘을 클릭한다. 그러면, 인터페이스가 좌측의 객체 리스트 pane 에 추가되면서 이름을 입력할 수 있게 될 것이다. 인터페이스에 적절한 이름을 쳐 넣는다. 이렇게 추가된 인터페이스는 디폴트로 설정된 속성을 가지게 되는데, 여기에서 적절한 속성 값들을 설정할 수 있다.
이제 인터페이스에 멤버 들을 추가할 차례이다. 우선 해당되는 인터페이스를 선택하고, 툴바에서 메소드를 추가할 때에는 메소드 아이콘을 프로퍼티를 추가할 때에는 프로퍼티 아이콘을 클릭한다.
추가된 멤버의 이름을 설정한다. 이들 멤버에는 기본적인 디폴트 설정 값이 Attributes 페이지에 나타나게 되는데, 이 내용을 멤버의 설계에 맞도록 수정한다.

물론, 이런 작업을 툴바를 클릭해서 하나하나 추가할 수도 있지만, Text 페이지에 파스칼 문법으로 직접 입력하는 것으로 이를 대신할 수도 있다.
일단 이런 식으로 멤버를 인터페이스에 추가하고 나면, 이들 멤버가 분리된 아이템으로 객체 리스트 pane(object list pane)에 나타날 것이다.
CoClass 역시 비슷한 방법으로 타입 라이브러리에 추가할 수 있다. 툴바에서 CoClass 아이콘을 클릭하고, 이름을 설정한 뒤에 Attributes 페이지에서 속성을 편집하면 된다. 여기에 멤버를 추가할 때에는 클래스가 지원할 인터페이스를 먼저 선택해야 하는데, Text 페이지에서 오른쪽 버튼을 클릭하면 선택가능한 인터페이스가 나열될 것이다. 여기에서 CoClass 가 구현한 인터페이스를 골라서 더블 클릭하면 적절하게 설정이 된다.
그 밖에 열거형이나 앨리어스, 레코드 등을 추가하는 방법에 대해서는 도움말을 참고하기 바란다.

- 타입 라이브러리의 배포

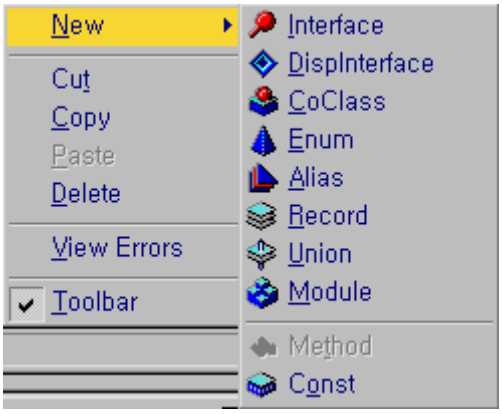
액티브 X 컨트롤 위저드를 이용해서 액티브 X 컨트롤을 제작할 경우, 타입 라이브러리는 자동으로 액티브 X 라이브러리 파일인 OCX 파일에 리소스로 포함되어 컴파일 된다. 그렇지만, 이를 분리된 .tlb 파일로 배포하는 것도 가능하다.
과거에는 자동화 어플리케이션의 경우 대부분 타입 라이브러리를 .tlb 파일에 저장하여 분리하여 배포하는 경우가 많았다. 그렇지만 델파이의 위저드를 사용할 경우 마찬가지로 .ocx, .exe 파일에 리소스로 포함되어 컴파일된다.
참고로 이런 형태로 컴파일 할 경우 액티브 X 라이브러리 하나에 여러 개의 타입 라이브러리가 포함될 수 있다. 독자적인 타입 라이브러리 파일을 배포하고자 한다면, 타입 라이브러리 에디터에서 이진 파일을 직접 저장할 수 있으므로 이를 이용할 수도 있다.
그렇지만, 쉽게 통합된 파일로 배포할 수 있는 것을 과거 방식대로 타입 라이브러리 파일을 분리해서 배포하는 것은 별로 권하고 싶지 않은 방법이다.

듀얼 인터페이스를 지원하는 자동화 서버의 제작

그러면, 듀얼 인터페이스를 지원하는 자동화 서버를 만들어 보자. 이번에는 액티브 X 라이브러리 DLL 파일이 아니라, 워드나 엑셀처럼 .exe 실행 파일로 된 out-of-process 서버로 만들어 보자.
먼저 New Application 메뉴를 선택해서 새로운 프로젝트를 연다. 그리고, 적당한 이름으로 저장을 한 뒤에 File|New 메뉴에서 ActiveX 탭의 Automation Object 아이콘을 더블 클릭한다. 나타나는 대화 상자에서 클래스 이름을 Sample 이라고 입력하고, 나머지 필드의 내용은 디폴트 값을 사용한다.

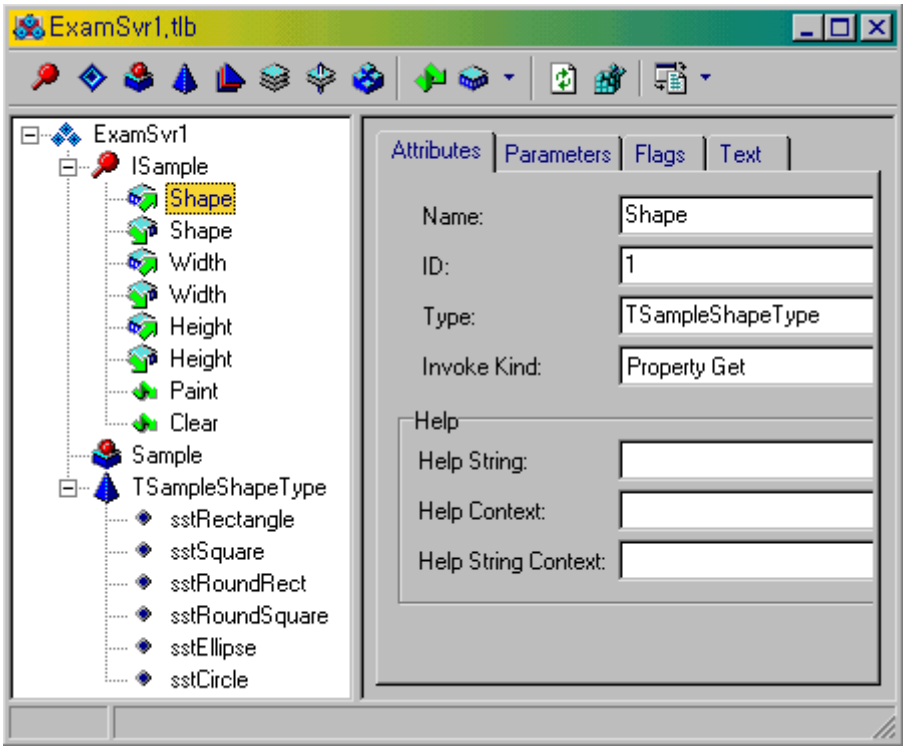
● 타입 라이브러리의 작성

OK 를 선택하면 타입 라이브러리 에디터가 뜨는데, 여기에는 이미 CoClass 인 Sample 과 ISample 인터페이스가 이미 정의되어 있을 것이다.
여기에 우리가 구상한 인터페이스의 멤버를 추가하도록 한다.
이번에 작성할 자동화 서버는 같이 포함된 프로젝트의 폼의 Canvas 를 이용하여 폼에 지정된 모양의 도형을 크기에 맞추어 그리도록 할 것이다.
그러므로 여기에서는 도형의 모양을 열거형으로 TSampleShape 을 선언하고, 이 데이터 형의 프로퍼티인 Shape 와 도형의 폭과 높이를 결정하는 Width, Height 프로퍼티를 멤버로 가진다. 그리고, 지정된 프로퍼티에 맞추어 그림을 그리는 Paint 메소드와 폼의 Canvas 를 지워주는 Clear 메소드로 구성한다.
그러면 타입 라이브러리에서 이런 요구 사항에 맞추어 인터페이스를 정의해보자. 먼저 열거형을 정의해야 한다. 타입 라이브러리 에디터에서 삼각뿔 모양의 아이콘인 New Enumeration 메뉴를 클릭한다. 그러면 새로운 열거형이 추가되는데, 열거형의 이름을 TSampleShape 으로 설정한다. 이제 TSampleShape 을 선택하고, 오른쪽 버튼을 클릭하면 팝업 메뉴가 생성되는데, 여기서 New 에 마우스 커서를 가져가면 다음과 같은 보조 메뉴가 나타날 것이다. 여기에서 Const 를 선택하여 멤버를 하나씩 입력하도록 한다.



TSampleShape 의 멤버로 sstRectangle, sstSquare, sstRoundRect, sstRoundSquare, sstEllipse, sstCircle 을 순서대로 설정한다.
이제 프로퍼티를 추가할 차례이다. 이 프로퍼티 들은 읽기와 쓰기가 모두 가능해야 하므로 ISample 인터페이스를 선택한 후 간단히 New Property 아이콘을 클릭하면 된다. 만약 읽기나 쓰기 전용 프로퍼티를 원할 경우에는 New Property 아이콘 옆의 화살표 아이콘을 클릭하여 해당되는 것을 선택할 수 있다. 먼저 Shape 프로퍼티를 추가하고 이름을 설정한 뒤, Type 콤보 박스에 TSampleShape 를 선택한다. 그리고, Width 와 Height 를 각각 추가하고 Type 콤보 박스에 Integer 를 선택한다. 이렇게 함으로써 프로퍼티 설정은 모두 끝났다. 메소드 역시 New Method 아이콘을 클릭하여 간단히 추가할 수 있다. 여기서 추가할

메소드인 Paint 와 Clear 는 모두 파라미터와 반환값이 없는 프로시저 형으로 선언할 것이다. 파라미터나 반환값에 대한 속성을 설정할 때에는 Parameters 탭에서 설정하면 된다.
이렇게 타입 라이브러리를 모두 설정하고 나면 다음 그림과 같은 형태로 타입 라이브러리 에디터가 보일 것이다.



- Out-of-process 자동화 서버의 제작

이제 프로젝트 파일을 저장(ExamSvr1.dpr, U_ExamSvr1.pas, U_ExamSvr1Impl.pas)하고 타입 라이브러리 에디터를 종료하면 구현할 유닛의 뼈대 코드가 만들어지는데, 여기에 uses 절에 구현에 필요한 루틴인 SysUtils, Graphics, Dialogs 를 추가하고 프로젝트의 폼을 이용할 것이므로 폼의 유닛 파일(여기서는 U_ExamSvr1)을 추가한다. 폼의 Width, Height 프로퍼티는 각 250 으로 설정하도록 하자. 그리고, 프로퍼티의 내용을 저장할 필드 변수인 FWidth, FHeight, FShape 를 private 섹션에 다음과 같이 선언하고, protected 섹션에 초기화를 위해 Initialize 메소드를 오버라이드하도록 한다.

```
unit U_ExamSvrImpl1;

interface

uses
```

```
  ComObj, ActiveX, SysUtils, Graphics, Dialogs, ExamSvr1_TLB, U_ExamSvr1;

type
  TSample = class(TAutoObject, ISample)
  private
    FShape: TSampleShapeType;
    FHeight: Integer;
    FWidth: Integer;
  protected
    function Get_Height: Integer; safecall;
    function Get_Shape: TSampleShapeType; safecall;
    function Get_Width: Integer; safecall;
    procedure Paint; safecall;
    procedure Set_Height(Value: Integer); safecall;
    procedure Set_Shape(Value: TSampleShapeType); safecall;
    procedure Set_Width(Value: Integer); safecall;
    procedure Initialize; override;
    procedure Clear; safecall;
  end;
```

그리고, initialization 섹션의 내용은 이미 다음과 같이 작성되어 있을 것이다.

```
initialization
  TAutoObjectFactory.Create(ComServer, TSample, Class_Sample,
    ciMultiInstance, tmApartment);
```

여기서 Ctrl+Shift+C 를 클릭하여 클래스 완료 메뉴를 동작시키면 선언된 메소드의 뼈대 코드가 자동으로 생성될 것이다. 이 뼈대 코드에 실제 구현 내용을 코딩하면 된다.
먼저 Shape, Width, Height 와 같은 프로퍼티의 구현 방법은 컴포넌트 제작할 때와 마찬가지로 Get_, Set_ 메소드를 이용해서 값을 읽거나 설정하도록 하면 된다.

```
function TSample.Get_Height: Integer;
begin
  Result := FHeight;
end;
```

```
function TSample.Get_Shape: TSampleShapeType;
begin
  Result := FShape;
end;

... (중략)

procedure TSample.Set_Width(Value: Integer);
begin
  FWidth := Value;
end;
```

Paint 와 Clear 메소드는 폼의 Canvas 를 이용하여 일반적인 드로우 메소드로 구현하면 된다. Paint 메소드는 지정된 크기의 도형을 서버 폼에 그려주는 역할을 하며, Clear 메소드는 이를 지우는 역할을 한다.

```
procedure TSample.Paint;
var
  X, Y, W, H, S: Integer;
begin
  with Form1.Canvas do
  begin
    Brush.Style := bsSolid;
    Brush.Color := clBlue;
    W := FWidth;
    H := FHeight;
    if W < H then S := W else S := H;
    case FShape of
      sstRectangle, sstRoundRect, sstEllipse:
        begin
          X := 0;
          Y := 0;
        end;
      sstSquare, sstRoundSquare, sstCircle:
        begin
          X := (W - S) div 2;
```

```
            Y := (H - S) div 2;
            W := S;
            H := S;
          end;
      end;
      case FShape of
        sstRectangle, sstSquare:
          Rectangle(X, Y, X + W, Y + H);
        sstRoundRect, sstRoundSquare:
          RoundRect(X, Y, X + W, Y + H, S div 4, S div 4);
        sstCircle, sstEllipse:
          Ellipse(X, Y, X + W, Y + H);
      end;
  end;
end;

procedure TSample.Clear;
begin
  with Form1.Canvas do
  begin
    Brush.Style := bsSolid;
    Brush.Color := clMenu;
    Rectangle(0, 0, Form1.Width, Form1.Height);
  end;
end;
```

초기화를 담당하는 Initialize 메소드는 컴포넌트의 constructor 의 역할을 하는 메소드이다. 여기서는 다음과 같이 기본적인 초기값을 설정하도록 한다.

```
procedure TSample.Initialize;
begin
  FWidth := 100;
  FHeight := 100;
  FShape := sstRectangle;
end;
```

이것으로 엑셀이나 엑세스와 같이 .exe 형태의 out-of-process 자동화 서버가 완성되었다. Out-of-process 자동화 서버는 그냥 .exe 파일을 배포하고, 사용자가 이를 실행할 때 /regserver 파라미터를 주는 것으로 간단히 레지스트리에 등록된다. 레지스트리에 등록되는 ProgID 는 일반적으로 실행파일의 이름과 CoClass 의 이름으로 이루어진다. 그러므로 여기서 작성한 예제의 경우 ExamSvr1.exe 파일이고, CoClass 는 Sample 이므로 'ExamSvr1.Sample'이 된다.

- In-process 서버의 제작

이렇게 배포와 등록이 간편한 장점이 있는 out-of-proc 자동화 서버의 단점은 in-proc 자동화 서버에 비해 프로세스간 주소의 마샬링 작업이 필요하기 때문에 수행성능이 떨어지는 단점이 있다.
그렇다면, 이와 똑같은 자동화 서버를 in-proc 서버로 작성해서 이들의 성능을 비교하는 클라이언트 어플리케이션을 제작해 보도록 하자.
먼저 File|New 메뉴의 ActiveX 탭에서 ActiveX Library 아이콘을 더블 클릭하여 액티브 X 서버 DLL 을 작성하기 위한 프로젝트 파일을 생성한다. 그리고, 지금까지의 방법과 마찬가지로 자동화 객체 위저드를 이용하여 Sample2 라는 새로운 자동화 서버를 작성하도록 하자. 타입 라이브러리에서 앞서 설명한 ISample 과 똑같은 과정을 거쳐서 멤버를 생성한다. 그런데, 같은 이름을 가지지 않도록 ISample2, TSampleShape2, Sample2 와 같이 뒤쪽에 2 를 붙이도록 한다. ISample 과 똑같이 인터페이스를 작성하고 타입 라이브러를 종료한 후 적당한 이름으로 프로젝트 파일을 저장(여기서는 ExamSvr2.dpr, U_ExamSvrImpl2.pas)하면 ExamSvr1 과 마찬가지로 코드가 생성될 것이다.
구현 방법이 거의 똑같기 때문에 자세한 코드 설명은 생략하고, 다르게 구현되는 부분만 중점적으로 설명하겠다.
처음에 DLL 로 작성했기 때문에, ExamSvr1 과는 달리 폼이 따로 존재하지 않으므로 새로 추가해야 한다. New Form 메뉴를 선택하여 프로젝트에 폼을 추가하고, 유닛의 이름을 U_ExamSvr2.pas 라는 이름으로 저장한다.
그리고, 자동화 서버가 시작할 때 폼이 생성되고 나타날 수 있도록 Initialize 메소드를 다음과 같이 오버라이드하여 작성한다.

```
procedure TSample2.Initialize;
begin
  FWidth := 100;
  FHeight := 100;
  FShape := sstRectangle;
  Form1 := TForm1.Create(Application);
```

```
  Form1.Width := 250;
  Form1.Height := 250;
  Form1.Show;
end;
```

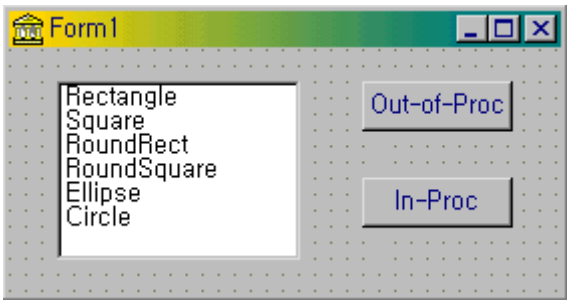
이를 위해서는 uses 절에 Forms.pas 유닛을 추가해야 한다. 또한, 자동화 서버가 종료할 때에는 폼이 파괴되어야 하므로, destructor 인 Destroy 프로시저 역시 다음과 같이 오버라이드하여야 한다.

```
destructor TSample2.Destroy;
begin
  Form1.Free;
  inherited Destroy;
end;
```

나머지 부분은 이름에 '2'를 추가하는 것 이외에는 모두 동일하므로 설명을 생략하도록 한다. 컴파일하고, 자동화 서버를 등록해야 하므로 Run|Register ActiveX Server 메뉴를 선택하여 자동화 서버를 사용할 수 있는 준비를 완료한다.

- 클라이언트 어플리케이션의 제작

이제 2 개의 동일한 역할을 하는 자동화 서버가 out-of-proc, in-proc 으로 모두 준비되었다. 그러면, 이들의 기능을 사용하는 클라이언트를 작성해보도록 하자.
먼저 New Application 메뉴를 선택한 후 다음과 같이 폼에 버튼 2 개와 라벨 2 개, 리스트 박스 1 개를 각각 올려 놓고, 버튼의 캡션을 각각 'Out-of-Proc', 'In-Proc'으로 설정한다. 라벨에는 백만분의 1 초 단위로 걸리는 시간을 나타낼 것이므로 캡션을 지우도록 한다.
또한, 리스트 박스에는 도형의 모양을 설정할 수 있도록 인터페이스의 TSampleShape 열거형의 순서에 맞도록 아이템을 설정한다. 이렇게 순서를 일치시키면 나중에 직접 ItemIndex 를 대입해도 문제가 없다(열거형은 0 부터 시작하는 일종의 정수형이다).



uses 절에 early 바인딩을 사용할 수 있도록 타입 라이브러리 파일(ExamSvr1_TLB.pas, ExamSvr2_TLB.pas)을 추가하고, ISample 과 ISample2 인터페이스를 담을 변수를 다음과 같이 전역 변수로 선언한다.

```
var
  Form1: TForm1;
  Sample: ISample;
  Sample2: ISample2;
```

폼의 OnCreate 이벤트 핸들러에서 이들 인터페이스를 생성하여 전역변수에 대입하고, 기본적으로 리스트 박스의 도형을 Rectangle 로 설정한다.

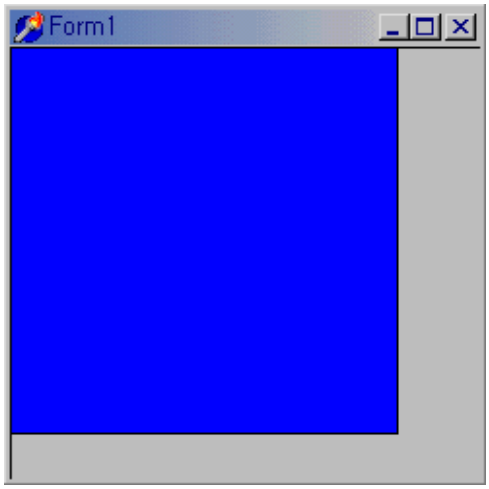
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.Selected[0];
  Sample := CoSample.Create;
  Sample2 := CoSample2.Create;
end;
```

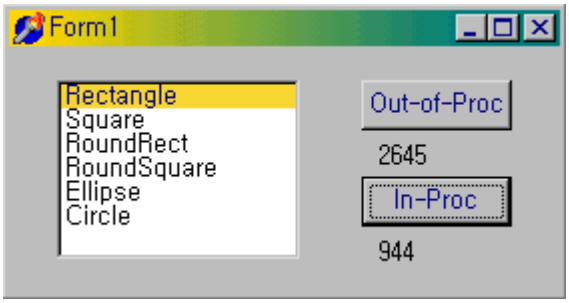
자, 그러면 이제 이들을 테스트하는 버튼의 OnClick 이벤트 핸들러를 작성할 차례가 되었다. 지난 장에서와 마찬가지로 GetTickCount 함수를 이용하여 시간을 잴 것이다.
크기를 폭과 높이를 동일하게 1~200 까지 증가시키면서 그릴 때 걸리는 시간을 비교하도록 다음과 같이 코딩한다.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i, StartTick, StopTick: Integer;
begin
  Sample.Shape := ListBox1.ItemIndex;
  StartTick := GetTickCount;
  for i := 1 to 200 do
  begin
    Sample.Clear;
    Sample.Width := i;
    Sample.Height := i;
```

```
    Sample.Paint;
  end;
  StopTick := GetTickCount;
  Label1.Caption := IntToStr(StopTick - StartTick);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  i, StartTick, StopTick: Integer;
begin
  Sample2.Shape := ListBox1.ItemIndex;
  StartTick := GetTickCount;
  for i := 1 to 200 do
  begin
    Sample2.Clear;
    Sample2.Width := i;
    Sample2.Height := i;
    Sample2.Paint;
  end;
  StopTick := GetTickCount;
  Label2.Caption := IntToStr(StopTick - StartTick);
end;
```

쉬운 내용이므로 따로 설명은 하지 않는다. 그러면, 이제 직접 실행을 해보자. 아마도 실행과 동시에 OnCreate 메소드에 의해 2 개의 서버 폼이 같이 뜰 것이다. 이들을 관찰할 수 있도록 위치를 옮긴 후, 버튼을 클릭하여 시간을 재보자 아마도 다음과 같은 결과를 보여줄 것이다.

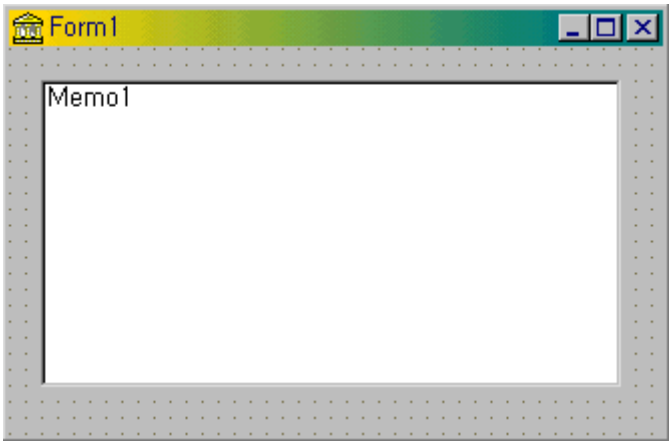




결과는 in-proc 서버가 2 배 이상 빠르다. 클라이언트 어플리케이션을 종료하면 자동화 서버 들은 자동으로 종료된다. 이런 속도 차이는 드로우와 같이 느린 작업을 한 결과로 나타난 것이므로, 계산이나 호출이 빈번한 경우에는 이보다 큰 차이가 날 것이다.

문자열 리스트, 폰트 등의 객체 활용

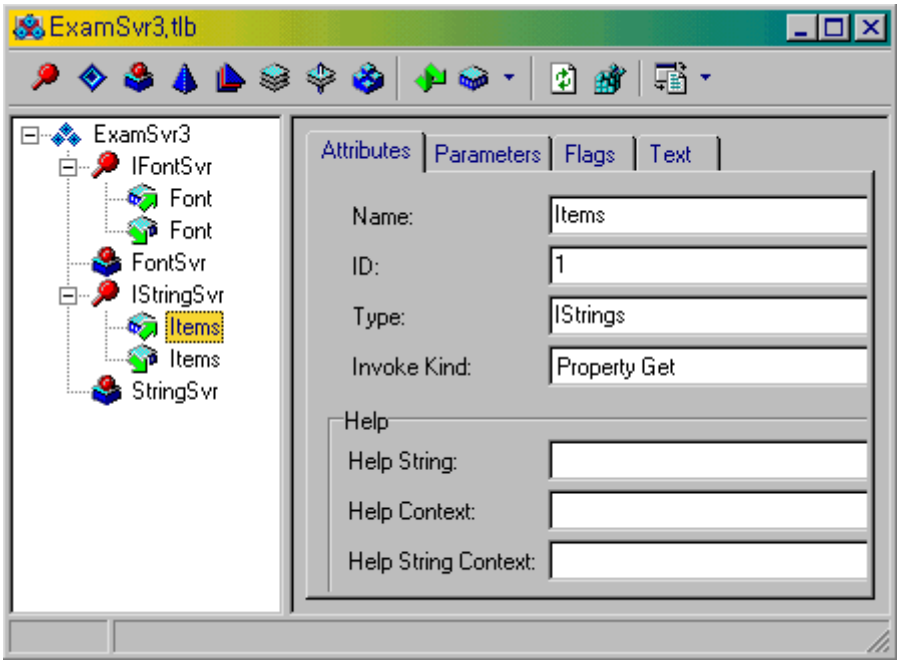
앞에서도 언급한 바 있지만, OLE 자동화를 이용할 때 주의할 점으로는 파라미터나 리턴 값으로 OLE 에 호환되는 데이터 형을 이용한다는 것이다.
그런데, 델파이의 문자열 리스트(string list), 폰트(font)나 그래픽 객체의 경우에는 여기에 해당되는 인터페이스를 이용해서 클라이언트와 서버 간의 통신이 가능하다. 물론 이 경우에는 지금까지 설명한 방법과는 다른 방법을 사용해야 하며, 델파이에서 지원하는 함수를 이용해야 한다.
그러면, 문자열 리스트와 폰트에 대한 정보를 주고받을 수 있는 자동화 서버와 클라이언트를 제작해 보도록 하자.
먼저 인터페이스로 폰트에 대한 인터페이스를 IFontSvr, 문자열 리스트에 대한 인터페이스를 IStringSvr 이라고 하고, 이들에 멤버로 프로퍼티인 Font 와 Items 를 각각 추가한다. 이들의 반환값으로는 IFontDisp 와 IStrings 를 사용할 것이다.
새로운 프로젝트를 시작하고, 폼에 다음과 같이 메모 컴포넌트를 하나 올려 놓는다. 메모 컴포넌트는 문자열 리스트를 클라이언트에서 받아서 보여 주고, 클라이언트로 메모 컴포넌트의 문자열 리스트를 보내는 등의 동작과 폰트의 정보를 반영하여 보여주기 위해서 올려 놓은 것으로 사실 클라이언트의 동작에는 영향을 미치지 않는다.



그리고, 필요한 자동화 객체가 CoFontSvr 과 CoStringSvr 의 2 개 이므로 2 번의 자동화 객체의 생성 작업이 필요하다. 먼저 IFontSvr 에 대한 작업을 하도록 하자. File|New 메뉴의 ActiveX 탭에서 Automation Object 아이콘을 더블 클릭한다. 클래스 이름으로

FontSvr 을 입력하고 OK 버튼을 클릭한다. 타입 라이브러리 에디터에서 IFontSvr 인터페이스를 선택하고, New Properties 버튼을 클릭하고 프로퍼티의 이름으로 Font 를 설정하고, Attributes 탭의 Type 콤보 박스에서 IFontDisp 를 선택한다. IFontDisp 가 TFont 클래스와 호화될 수 있는 역할을 하는 인터페이스이다.

타입 라이브러리 에디터를 일단 닫고 프로젝트 파일을 저장한 후, 다시 한번 File|New 메뉴를 선택한 뒤 Automation Object 아이콘을 더블 클릭하여 StringSvr 클래스를 추가한다. 같은 타입 라이브러리 에디터가 나타나면 IStringSvr 인터페이스를 선택하고, New Properties 버튼을 클릭한 후 프로퍼티의 이름을 Items 로 설정한다. Attributes 탭의 Type 콤보 박스에서 IStrings 를 선택한다. 이렇게 한 뒤의 타입 라이브러리 에디터의 형태는 다음 그림과 같을 것이다.



이렇게 하면 자동화 서버를 구현하는 유닛이 2 개가 추가되었을 것이다. 하나는 FontSvr 에 대한 것이고, 나머지 하나는 StringSvr 에 대한 것이다. 먼저 FontSvr 클래스를 구현하도록 하자. TFontSvr 클래스가 이미 만들어져 있겠지만, 다음과 같이 uses 절에 구현에 필요한 AxCtrls, Graphics, StdVCL 유닛과 ExamSvr3_TLB 유닛, 그리고 폼을 이용할 것이므로 폼의 유닛인 U_ExamSvr3 를 추가하고 선언부를 다음과 같이 수정하도록 한다.

```
unit U_ExamSvrImpl3;
interface
uses
  ComObj, ActiveX, StdVCL, AxCtrls, Graphics, ExamSvr3_TLB, U_ExamSvr3;
```

```
type
  TFontSvr = class(TAutoObject, IFontSvr)
  private
    FFont: TFont;
  public
    procedure Initialize; override;
    destructor Destroy; override;
    function Get_Font: IFontDisp; safecall;
    procedure Set_Font(const Value: IFontDisp); safecall;
  end;
```

FFont 필드에 델파이의 TFont 에 해당되는 내용을 저장하고, 이를 IFontDisp 인터페이스에 맞도록 변환하는 처리를 Get_Font, Set_Font 메소드에서 하게 된다.
일단 Initialize 프로시저와 Destroy 프로시저를 다음과 같이 구현하여 폰트 객체를 생성하고, 이를 해제한다.

```
procedure TFontSvr.Initialize;
begin
  inherited Initialize;
  FFont := TFont.Create;
end;

destructor TFontSvr.Destroy;
begin
  FFont.Free;
  inherited Destroy;
end;
```

그리고, 핵심이 되는 Get_Font, Set_Font 메소드를 다음과 같이 구현하면 된다.

```
function TFontSvr.Get_Font: IFontDisp;
begin
  FFont.Assign(Form1.Memo1.Font);
  GetOleFont(FFont, Result);
end;
```

```
procedure TFontSvr.Set_Font(const Value: IFontDisp);
begin
  SetOleFont(FFont, Value);
  Form1.Memo1.Font.Assign(FFont);
end;
```

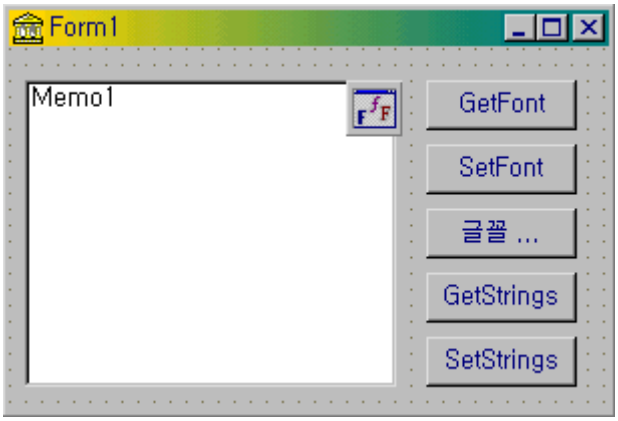
구현의 핵심이 되는 것이 GetOleFont, SetOleFont 함수이다. 이들 함수는 델파이의 AxCtrls.pas 유닛에 포함된 것으로 델파이의 TFont 와 IFontDisp 인터페이스를 서로 변환할 수 있도록 지원하는 역할을 한다.
마찬가지로 IStrings 나 IPicture 를 지원하는 GetOleStrings, SetOleStrings, GetOlePicture, SetOlePicture 함수를 이용할 수 있다.
TStringSvr 클래스의 구현 방법도 대동소이하다. 먼저, uses 절에서 Graphics 대신에 Classes 유닛을 추가하고 다음과 같이 선언한다.

```
unit U2_ExamSvrImpl3;
interface
uses
  ComObj, ActiveX, StdVCL, Classes, AxCtrls, ExamSvr3_TLB, U_ExamSvr3;

type
  TStringSvr = class(TAutoObject, IStringSvr)
  private
    FItems: TStrings;
  public
    procedure Initialize; override;
    function Get_Items: IStrings; safecall;
    procedure Set_Items(const Value: IStrings); safecall;
  end;
```

Destroy 메소드에 대한 선언부가 빠진 이유는 어떤 이유인지는 잘 모르겠으나, 생성한 TStringList 객체를 파괴하는 메소드를 추가하면, 상속된 Destroy 메소드에서 이를 파괴하기 위해 다시 접근하는 탓에 Access violation 에러가 발생한다. 그래서, 이 부분을 삭제하니 아무런 문제가 발생하지 않았다.
이들의 구현 방법은 다음과 같다.

```
procedure TStringSvr.Initialize;
begin
  inherited Initialize;
  FItems := TStringList.Create;
end;

function TStringSvr.Get_Items: IStrings;
begin
  FItems := Form1.Memo1.Lines;
  GetOleStrings(FItems, Result);
end;

procedure TStringSvr.Set_Items(const Value: IStrings);
begin
  SetOleStrings(FItems, Value);
  Form1.Memo1.Lines.Assign(FItems);
end;
```

역시 핵심은 GetOleStrings 와 SetOleStrings 함수를 이용하여 IStrings 와 TStrings 간의 변환을 하는 부분이다. 이제 프로젝트를 컴파일하고 /regserver 옵션을 주고 실행하면 서버를 사용할 수 있게 된다.
그러면, 이제 이들을 이용하는 클라이언트 어플리케이션을 만들어 보자
새로운 프로젝트를 선택하고, 폼에 메모 컴포넌트 1 개와 버튼을 5 개와 TFontDialog 컴포넌트를 하나 올려 놓고 다음과 같이 캡션을 설정하도록 한다.



여기서 GetFont 버튼은 IFontSvr 의 폰트 정보를 메모 컴포넌트에 적용하고, SetFont 버튼

은 IFontSvr 의 Font 프로퍼티에 메모 컴포넌트의 폰트를 저장한다. '글꼴...' 버튼은 메모 컴포넌트의 폰트를 변경하기 위한 목적으로 사용된다. 마찬가지로 GetStrings 버튼은 서버의 Items 프로퍼티에 저장된 문자열을 메모 컴포넌트로 불러오게 되며, SetStrings 버튼은 메모 컴포넌트의 문자열을 IStringSvr 의 Items 프로퍼티에 저장하는 역할을 한다.
먼저, IStringSvr 인터페이스와 IFontSvr 인터페이스, IFontDisp 와 IStrings 인터페이스를 저장할 전역 변수를 다음과 같이 선언하고, 폼의 OnCreate 이벤트 핸들러를 다음과 같이 작성하여 이들을 early 바인딩을 이용해서 대입한다.

```
var
  Form1: TForm1;
  FontSvr: IFontSvr;
  StringSvr: IStringSvr;
  FFont: IFontDisp;
  FItems: IStrings;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Memo1.Lines.Clear;
  FontSvr := CoFontSvr.Create;
  StringSvr := CoStringSvr.Create;
end;
```

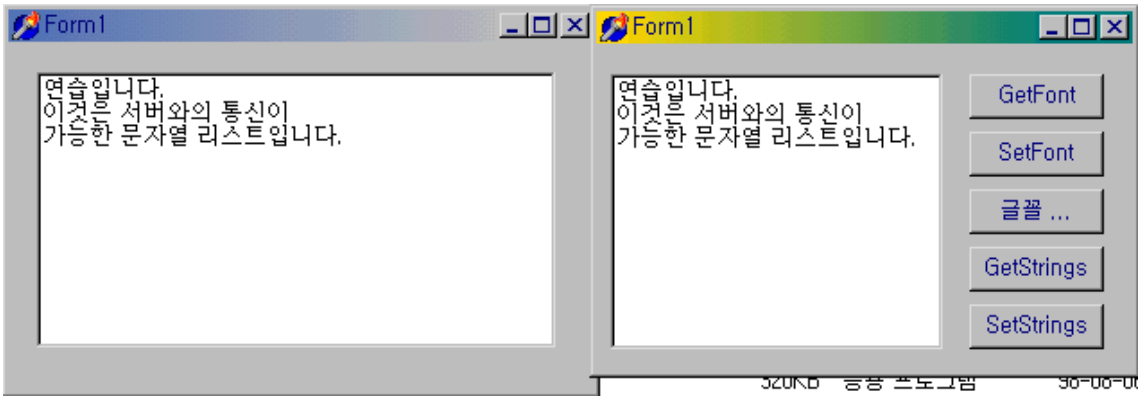
물론 여기서 FFont 와 FItems 변수의 경우 임시 변수로 사용되는 역할을 하기 때문에 전역 변수로 선언하지 않고, 버튼의 OnClick 이벤트 핸들러에서 지역 변수로 선언해서 사용하는 것이 더 효율적인 사용 방법이다. 그렇지만, 이 예제에서 전역 변수로 선언한 이유는 예제 코드의 중복을 없애서 길이를 줄이고자 한 것이므로 이를 고려하기 바란다.
마지막으로, 앞에서 설명한 기능을 하는 5 개 버튼의 OnClick 이벤트 핸들러를 다음과 같이 작성하면 된다.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  FFont := FontSvr.Font;
  SetOleFont(Memo1.Font, FFont);
end;

procedure TForm1.Button2Click(Sender: TObject);
```

```
begin
  GetOleFont(Memo1.Font, FFont);
  FontSvr.Font := FFont;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
  if FontDialog1.Execute then Memo1.Font := FontDialog1.Font;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  FItems := StringSvr.Items;
  SetOleStrings(Memo1.Lines, FItems);
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  GetOleStrings(Memo1.Lines, FItems);
  StringSvr.Items := FItems;
end;
```

당연히 이들 코드를 사용하기 위해서는 GetOleStrings 등의 함수가 선언된 AxCtrls.pas 유닛과 표준 인터페이스의 선언부가 위치한 StdVCL.pas 유닛, 그리고 IStringSvr 과 IFontSvr 인터페이스가 선언되어 있는 ExamSvr3_TLB.pas 유닛을 uses 절에 추가해야 한다.

그러면, 클라이언트를 실행하고 폰트와 메모 컴포넌트의 내용을 바꿔 가면서 서버와의 통신을 해보기 바란다. 다음과 같이 서버와 클라이언트 간의 문자열 리스트와 폰트의 정보를 쉽게 주고 받는 과정을 눈으로 확인할 수 있을 것이다.



정 리 (Summary)

이번 장에서는 기본적인 자동화 서버를 작성하는 방법과 타입 라이브러리 에디터를 이용하는 방법에 대해서 알아보았다. 자동화 서버는 윈도우 95 를 지원하는 어플리케이션을 제작할 때 사용자에게 많은 편리성을 제공할 수 있는 도구가 된다.

또한, 공통적으로 사용하는 기능이 있을 경우 이를 자동화 서버로 구현해 놓으면 개발 도구를 가리지 않고, 어느 곳에서나 사용할 수 있으므로 그 활용도가 매우 높다.

다음 장에서는 액티브 X 컨트롤과 액티브 폼을 제작하고, 이를 활용하는 방법에 대해서 알아볼 것이다.