ТЕХНОЛОГИЯ КОМПОНЕНТНОГО ПРОГРАММИРОВАНИЯ

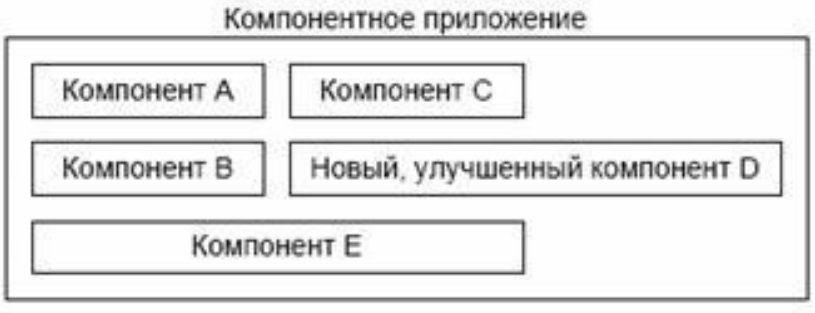
ВВЕДЕНИЕ

Традиционно разрабатываемое приложение формируется из отдельных файлов, модулей или классов, которые компилируются и компонуются в единое целое. После того, как приложение сгенерировано компилятором, он остается неизменным до тех пор, пока не будет скомпилирована его новая версия.

С каждым новым этапом развития технологий программирования подобный подход создания приложений все сильнее устаревает. В связи с этим перед разработчиками постоянно стоит задача найти способ вдохнуть новую жизнь в программы, которые уже поставлены пользователям.

Решение состоит в том, чтобы разбить монолитное приложение на отдельные части – **компоненты**.

По мере развития технологии компоненты, составляющие приложение, могут заменяться новыми. Это позволяет делать приложения не статичными, а постепенно эволюционирующими – в результате замены старых компонентов новыми. Подобный подход также позволяет из существующих компонентов легко создавать и абсолютно новые приложения.



Разработка приложений из компонентов – так называемых «**приложений компонентной архитектуры**» достаточно новая технология. Компонент подобен миниприложению: он поставляется пользователю как двоичный код, скомпилированный и готовый к использованию. Единого целого больше нет. Его место занимают специализированные компоненты, которые подключаются во время выполнения к другим компонентам, формируя приложение. Модификация или расширение приложения сводится к простой замене одного из составляющих его компонентов новой версией.

Принято считать, что основополагающей технологией разбиения приложений на компоненты является технология COM.

Component Object Model (COM) или модель компонентных объектов – это спецификация метода создания компонентов и построения из них приложений. COM была разработана в середине 80-х годов 20-го века компанией Microsoft для того, чтобы сделать программные продукты этой фирмы более гибкими, динамичными и настраиваемыми. Практически все продаваемые сегодня приложения Microsoft используют COM. Однако существуют и разработки других фирм, ориентированные на построение приложений с компонентной архитектурой. Среди и тех и других известны следующие: COM, DCOM, OLE, CORBA, Java, .NET и др.

Технология COM

Основу технологии COM составляет спецификация, которая указывает, как создавать динамически взаимозаменяемые компоненты. **COM определяет стандарт**, которому должны следовать компоненты и клиенты, чтобы гарантировать возможность **совместной работы**.

Компоненты COM состоят из исполняемого кода, распространяемого в виде динамически компонуемых библиотек (DLL) или EXE-файлов Win32.

Для динамического подключения друг к другу компоненты COM используют DLL. Однако, сама по себе динамическая компоновка не обеспечивает компонентной архитектуры: компоненты должны быть инкапсулированы.

Обеспечение инкапсуляции в компонентах COM достигается **за счет** соблюдения ряда следующих технологических ограничений:

1. Компоненты полностью независимы от языка программирования (т.е. могут быть разработаны с помощью практически любого процедурного языка).

2. Компоненты могут распространяться в информационной среде в двоичной форме.

3. Компоненты можно модернизировать, не нарушая работы старых клиентов (COM предоставляет стандартный способ реализации разных версий компонента).

4. Компоненты COM можно прозрачно перемещать по сети (компонент на удаленной системе рассматривается так же, как компонент на локальном диске компьютера).

ОС Windows выстроена на основе технологии COM. COM-объекты выполняют разные функции, создаются и уничтожаются системой автоматически (по мере возникновения запросов от пользователей). Достаточно только описать интерфейс

COM-объекта и запрограммировать его реализацию, все остальное выполняет компилятор и Windows.

При разработке приложений COM используют следующие составные части технологии:

Интерфейс COM – описывает методы и свойства, доступные программам, обращающимся к объекту.

Сервер COM – законченный модуль кода (EXE или DLL), в котором хранится программный код одного или нескольких объектов COM.

Клиент COM – программный код, в котором происходит обращение к интерфейсу с запросом на выполнение услуг сервера COM.

Интерфейсы составляют основу данной технологии. Для клиента компонент представляет собой набор интерфейсов, которые являются единственным каналом взаимодействия клиента с компонентом COM.

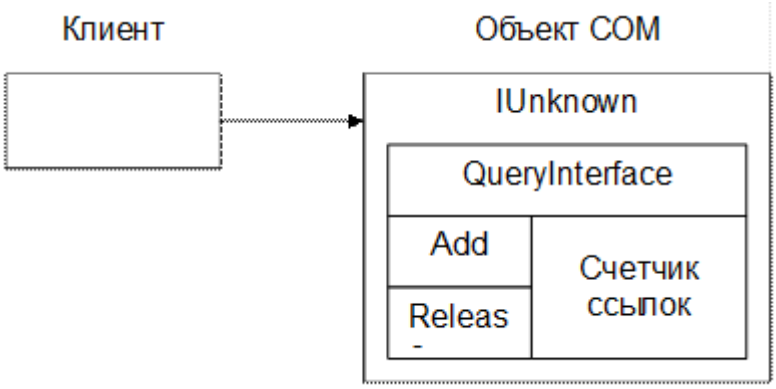
Компонент можно удалить и заменить другим; если новый компонент поддерживает те же интерфейсы, что и старый, приложение будет работать по-прежнему.

Интерфейс COM позволяет клиентам COM общаться с сервером на основе стандартного механизма публикации интерфейса. После того, как **интерфейс COM опубликова**н (стандартным способом зарегистрирован системой), изменять его нельзя, что гарантирует одинаковую работу объекта COM в любых условиях.

У любого объекта COM имеется базовый интерфейс – IUnknown, который позволяет узнать, какие еще интерфейсы COM доступны для клиента COM. Все эти интерфейсы наследуют характеристики интерфейса IUnknown.

Уникальность интерфейса обеспечивается его глобальным идентификатором – Globally Unique IDentifier (**GUID**) длиной 16 байтов. Каждый объект COM имеет идентификатор интерфейса – Interface IDentifier (**IID**) на основе GUID. GUID требуется для избежания проблем при появлении интерфейсов COM с одинаковыми именами.

Благодаря наличию стандартных интерфейсов объект **COM может быть реализован на любом языке программирования**. Интерфейс IUnknown содержит метод QueryInterface, возвращающий ссылку на другие доступные интерфейсы, а также методы AddRef и Release, которые увеличивают и уменьшают счетчик ссылок на конкретный интерфейс.



Счетчик увеличивается при обращении к интерфейсу и уменьшается при освобождении интерфейса. Как только значение счетчика равно нулю, т.е. к интерфейсу больше нет обращений, соответствующий объект COM может быть удален из памяти до следующего запроса к его интерфейсу.

На основе технологии COM был создан ряд расширений: серверы автоматизации (OLE), активные серверные страницы (.ASP), встраиваемые серверы ActiveX с визуальной настройкой и др.

1. Суть и содержание OLE

Технология OLE

Сервер COM

При обращении к серверу клиент COM передает ему идентификатор класса. **Идентификатор класса** (**CLSID**) – это GUID, который ссылается на подходящий объект COM.

Сервер COM создает специальный объект – фабрику классов (IClassFactory), который занимается непосредственно созданием и загрузкой (производством) экземпляра нужного объекта COM, выполняющего конкретные действия его интерфейса, указанные в запросе клиента COM.

Фабрика классов «выпускает» объект COM, реализующий один или несколько интерфейсов COM, а также экземпляр специального класса ComClass, который обеспечивает возможность обращения к объекту COM на основе интерфейсов COM.

Сервер COM может быть реализован тремя способами:

1. В виде библиотеки .DLL. При этом объект COM выполняется в адресном пространстве обратившегося к нему приложения.

2. В виде приложения .EXE, которое выполняется в собственном адресном пространстве, но на одной машине с клиентом COM.

3. В виде библиотеки .DLL или приложения .EXE, которые загружаются и работают на иной машине, нежели клиент COM (технология **DCOM**).

Подход, при котором разделение компонентов происходит на уровне самостоятельных приложений, получил название Object Linking and Embedding (OLE). Он был положен корпорацией Microsoft в основу связывания и внедрения объектов, и сегодня OLE – это огромная и сложная технология, которая оказывает определяющее влияние на отдельные стили программирования.

В общем случае технология OLE предназначена для объединения инструментальных возможностей разных приложений независимо от их форматов данных, операционной среды, языка программирования и разработчика. Она поддерживает внедрение копии одного компонента в «контейнер» (в другой компонент), включение в компонент ссылки на другой компонент (связывание компонентов) и сохранение составных объектов (документов).

Однако связыванием и внедрением OLE занималась только при рождении своего названия. В настоящее время **OLE** разрослась и **включает целую группу технологий**: связывание, внедрение, перетаскивание (drag-and-drop), буфер обмена (clipboard), структурированную память, автоматизацию и др.

Все они базируются на программной технологии COM, которая в свою очередь, выстроена на основе концепции объектно-ориентированного системного обеспечения – Object Oriented System Software (**OOSS**).

Объект в технологии OLE – это информационный элемент, объединяемый методами OLE с другими данными. Такой объект – это не то же, что объект в ООП. Поэтому точнее его можно называть как **OLE-объект**. OLE-объектом может быть целый документ (файл), отдельный его фрагмент или даже отдельный символ.

Автоматизация OLE – это протокол, посредством которого одно приложение может получить доступ к объекту, размещённому внутри другого приложения или DLL. Доступ к этому объекту предоставляет возможность:

- управлять действиями приложения или DLL;

- получить доступ к свойствам приложения или DLL.

Приложение, которое может быть автоматизировано, называется **сервером автоматизации**. Приложение, которое автоматизирует другое приложение, называется **клиентом автоматизации**. При этом приложение может быть одновременно как сервером, так и клиентом.

Автоматизация существенно увеличивает интеграцию программного обеспечения. Классический пример автоматизации – это пакет Microsoft Office, в котором можно управлять из одного приложения другими приложениями.

Построение серверов и клиентов автоматизации поддерживается в среде Delphi. Это означает, что можно использовать приложение Delphi для автоматизации другого приложения, или можно установить приложение так, чтобы оно было доступно для автоматизации другими приложениями.

Автоматизация – это идеология доступа к объектам приложения и манипуляция с ними извне. **Автоматные объекты** доступны только программно, т.е. программа-клиент даёт команды программе серверу, и та выполняет эти команды, предоставляя программе-клиенту результат. Команды могут писаться на любых языках или макроязыках. Автоматные объекты не видны конечному пользователю и используются обычно для автоматизации часто повторяемых задач.

Автоматные объекты не могут связываться или внедряться в прикладное приложение. Они временны, существуют только во время выполнения приложения и доступны только с помощью заранее запрограммированного дистанционного управления.

Автоматизация требует три вида информации:

1. Класс OLE-объекта (например, Excel, Word, Matlab и т.д.)

2. Документ OLE – файл с данными объекта.

3. Элемент OLE – часть документа, подлежащая связыванию или внедрению.

Существует два основных типа автоматизации:

- серверы автоматизации;

- клиенты автоматизации.

Сервер автоматизации – это приложение или DLL, которое владеет объектом. **Клиент автоматизации** – приложение или DLL, которое получает доступ к объекту. Одно приложение или DLL может выступать и в качестве сервера, и в качестве клиента. При этом автоматизация определяет **два вида серверов автоматизации OLE**:

- серверы внутренней обработки;

- локальные серверы.

Серверы внутренней обработки – это DLL, которые загружаются в адресное пространство прикладной программы. Причина для создания серверов внутренней обработки – совместное использование объекта, написанного на одном языке, и приложения, написанного на другом языке. Можно, к примеру, совместно использовать объекты Delphi, приложения C++ и Visual Basic.

Локальные серверы – это автономные исполняемые модули, которые содержат серверы автоматизации. Классический пример локального сервера – Excel, Word.

При отсутствии необходимости пересекать языковые барьеры, DLL в использовании быстрее и проще.

2. Основы технологии COM в Delphi

Одним из главных достоинств Delphi является поддержка технологий COM и ActiveX. В этой главе мы рассмотрим, что представляет собой технология COM, в чем различие между технологиями COM, ActiveX и OLE.

Итак, *COM* (Component Object Model) – это объектная модель компонентов. Данная технология является базовой для технологий ActiveX и OLE. Технологии OLE и ActiveX – всего лишь надстройки над данной технологией. В качестве примера можно привести объект TObject, как базовый объект VCL Delphi. Точно так же технология COM является базовой по отношению к OLE и ActiveX.

Технология COM применяется при описании API и двоичного стандарта для связи объектов различных языков и сред программирования. COM предоставляет модель взаимодействия между компонентами и приложениями.

Технология COM работает с так называемыми COM-объектами. COM-объекты похожи на обычные объекты визуальной библиотеки компонентов Delphi. В отличие от объектов VCL Delphi, COM-объекты содержат свойства, методы и интерфейсы.

Обычный COM-объект включает в себя один или несколько интерфейсов. Каждый из этих интерфейсов имеет собственный указатель.

Технология COM имеет два явных плюса:

– создание COM-объектов не зависит от языка программирования. Таким образом, COM-объекты могут быть написаны на различных языках;

– COM-объекты могут быть использованы в любой среде программирования под Windows. В число этих сред входят Delphi, Visual C++, C++Builder, Visual Basic, и многие другие.

Примечание

Хотя технология COM обладает явными плюсами, она имеет также и минусы, среди которых зависимость от платформы. То есть, данная технология применима только в операционной системе Windows и на платформе Intel.

Все COM-объекты обычно содержатся в файлах с расширением DLL или OCX. Один такой файл может содержать как одиночный COM-объект, так и несколько COM-объектов.

Ключевым аспектом технологии COM является возможность предоставления связи и взаимодействия между компонентами и приложениями, а также реализация клиент-серверных взаимодействий при помощи интерфейсов.

Технология COM реализуется с помощью *COM-библиотек* (в число которых входят такие файлы операционной системы, как OLE32.DLL и OLE-Aut32.DLL). COM-библиотеки содержат набор стандартных интерфейсов, которые обеспечивают функциональность COM-объекта, а также небольшой набор функций API, отвечающих за создание и управление COM-объектов.

В Delphi реализация и поддержка технологии COM называется *каркасом Delphi ActiveX* (Delphi ActiveX FrameWork, DAX). Реализация DAX описана в модуле AxCtrls.

Развитие COM-технологий

Одной из важнейших задач, которые ставила перед собой фирма Microsoft, когда продвигала операционную систему Windows, была задача по обеспечению эффективного взаимодействия между различными программами, работающими в Windows.

Самыми первыми попытками решить эту непростую задачу были буфер обмена, разделяемые файлы и технология *динамического обмена данными* (Dynamic Data Exchange, DDE).

После чего была разработана технология связывания и внедрения объектов (Object Linking and Embedding – OLE). Первоначальная версия OLE 1 предназначалась для создания составных документов. Эта версия была признана несовершенной и на смену ей пришла версия OLE 2. Новая версия позволяла решить вопросы предоставления друг другу различными программами собственных функций. Данная технология активно внедрялась до 1996 года, после чего ей на смену пришла технология ActiveX, которая включает в себя автоматизацию (OLE-автоматизацию), контейнеры, управляющие элементы, Web-технологию и т. д.

Терминология COM

Всякая новая технология приносит с собой новые термины для ее описания.

COM-объект

COM-объект представляет собой двоичный код, который выполняет какую-либо функцию и имеет один или более интерфейс.

COM-объект содержит методы, которые позволяют приложению пользоваться COM-объектом. Эти методы доступны благодаря COM-интерфейсам. Клиенту достаточно знать несколько базовых интерфейсов COM-объекта, чтобы получить полную информацию о составе свойств и методов объекта. COM-объект может содержать один или несколько интерфейсов. Для программиста COM-объект работает так же, как и класс в Object Pascal.

COM-интерфейсы

COM-интерфейс применяется для объединения методов COM-объекта. Интерфейс позволяет клиенту правильно обратиться к COM-объекту, а объекту – правильно ответить клиенту. Названия COM-интерфейсов начинаются с буквы I. Клиент может не знать, какие интерфейсы имеются у COM-объекта. Для того чтобы получить их список, клиент использует базовый интерфейс IUnknown, который есть у каждого COM-объекта.

Пользователь COM-объекта

Пользователем COM-объекта называется приложение или часть приложения, которое использует COM-объект и его интерфейсы в своих собственных целях. Как правило, COM-объект находится в другом приложении.

COM-классы

COM со-классы (coclass) – это классы, которые содержат один или более COM-интерфейс. Вы можете не обращаться к COM-интерфейсу непосредственно, а получать доступ к COM-интерфейсу через со-класс. Со-классы идентифицируются при помощи идентификаторов класса (CLSID).

Библиотеки типов

COM-объекты часто используют библиотеки типов. *Библиотека типов* – это специальный файл, который содержит информацию о COM-объекте. Данная информация содержит список свойств, методов, интерфейсов, структур и других элементов, которые содержатся в COM-объекте. Библиотека типов содержит также информацию о типах данных каждого свойства и Типах данных, возвращаемых методами COM-объекта.

Файлы библиотеки типов имеют расширение TLB.

Технология DCOM

Технология DCOM (Distributed COM) – это распределенная COM-технология. Она применяется для предоставления средств доступа к COM-объектам, расположенным на других компьютерах в сети (в том числе и сети Internet).

Операционные системы Windows имеют встроенную поддержку DCOM.

Счетчики ссылок

Каждый COM-объект имеет счетчик ссылок. Данный счетчик содержит число процессов, которые в текущий момент времени используют COM-объект. Под процессом здесь подразумевается любое приложение или DLL, которые используют COM-объект, т. е. пользователи COM-объекта. Счетчик ссылок на COM-объект нужен для того, чтобы высвобождать процессорное время и оперативную *память, занимаемую COM-объектом, в том* случае, когда он не используется.

После создания и обращения к COM-объекту счетчик ссылок увеличивается на единицу. Всякий раз, когда новое приложение подключается к COM-объекту – счетчик увеличивается. Когда процесс отключается от COM-объекта – счетчик уменьшается. При достижении счетчиком нуля память, занимаемая COM-объектом, высвобождается.

OLE-объекты

Часть данных, использующаяся совместно несколькими приложениями, называется *OLE-объектом.* Те приложения, которые могут содержать в себе OLE-объекты, называются *OLE-контейнерами* (OLE container). Приложения, имеющие возможность содержать свои данные в OLE-контейнерах, называются *OLE-серверами* (OLE server).

Составные документы

Документ, включающий в себя один или несколько OLE-объектов, называется *составным документом.* Приложение, которое может содержаться внутри документа, называется *ActiveX-документом* (ActiveX document).

Остальные термины, присущие технологии COM, мы рассмотрим в следующих разделах данной книги.

Состав COM-приложения

При создании COM-приложения необходимо обеспечить следующее:

- COM-интерфейс;

- COM-сервер;

- COM-клиент.

Рассмотрим эти три составляющие COM-приложения более подробно.

COM-интерфейс

Клиенты COM связываются с объектами при помощи COM-интерфейсов. *Интерфейсы* – это группы логически или семантически связанных процедур, которые обеспечивают связь между поставщиком услуги (сервером) и его клиентом.

Для примера, каждый COM-объект всегда поддерживает основной COM-интерфейс IUnknown, который применяется для передачи клиенту сведений о поддерживаемых интерфейсах.

Как уже говорилось выше, COM-объект может иметь несколько интерфейсов, каждый из которых обеспечивает какую-либо свою функцию.

Ключевыми аспектами COM-интерфейсов являются следующие.

– Однажды определенные, интерфейсы не могут быть изменены. Таким образом, вы можете возложить на один интерфейс определенный набор функций. Дополнительную функциональность можно реализовать с помощью дополнительных интерфейсов.

– По взаимному соглашению, все имена интерфейсов начинаются с буквы *I*, например IPersist, IMalloc.

– Каждый интерфейс гарантированно имеет свой уникальный идентификатор, который называется *глобальный уникальный идентификатор* (Globally Unique Identifier, GUID). Уникальные идентификаторы интерфейсов называют *идентификаторами интерфейсов* (Interface Identifiers, IIDs). Данные идентификаторы обеспечивают устранение конфликтов имен различных версий приложения или разных приложений.

– Интерфейсы не зависят от языка программирования. Вы можете воспользоваться любым языком программирования для реализации COM-интерфейса. Язык программирования должен поддерживать структуру указателей, а также иметь возможность вызова функции при помощи указателя явно или неявно.

– Интерфейсы не являются самостоятельными объектами, они лишь обеспечивают доступ к объектам. Таким образом, клиенты не могут напрямую обращаться к данным, доступ осуществляется при помощи указателей интерфейсов.

– Все интерфейсы всегда являются потомками базового интерфейса IUnknown.

Основной COM-интерфейс *IUnknown*

Интерфейс IUnknown обеспечивает механизм учета ссылок (счетчик ссылок на COM-объект). При передаче указателя на интерфейс выполняется метод интерфейса IUnknown AddRef. По завершении работы с интерфейсом приложение-клиент вызывает метод Release, который уменьшает счетчик ссылок.

При вызове метода QueryInterface интерфейса IUnknown в метод передается параметр IID, имеющий тип TGUID, т. е. идентификатор интерфейса. Параметр метода out возвращает либо ссылку на запрашиваемый интерфейс, либо значение NH. Результатом вызова метода может быть одно из значений, перечисленных в табл. 2.1.

Таблица 2.1. Значения, возвращаемые методом QueryInterface

Значение	Описание
S_OK	Интерфейс поддерживается
E_NOINTERFACE	Интерфейс не поддерживается
E_UNEXPECTED	Неизвестная ошибка

Указатели COM-интерфейса

Указатель интерфейса – это 32-битный указатель на экземпляр объекта, который является, в свою очередь, указателем на реализацию каждого метода интерфейса. Реализация методов доступна через массив указателей на эти методы, который называется vtable. Использование массива vtable похоже на механизм поддержки виртуальных функций в Object Pascal.

COM-серверы

Клиенты не знают, *как* COM-объект выполняет свои действия. COM-объект предоставляет свои услуги при помощи интерфейсов. В дополнение, приложению-клиенту не нужно знать, где находится COM-объект. Технология COM обеспечивает прозрачный доступ независимо от местонахождения COM-объекта.

В общих чертах, COM-сервер должен выполнять следующее:

– регистрировать данные в системном реестре Windows для связывания модуля сервера с идентификатором класса (CLSID);

– предоставлять фабрику COM-класса, создающую экземпляры COM-объектов;

– обеспечивать механизм, который выгружает из памяти серверы COM, которые в текущий момент времени не предоставляют услуг клиентам.

Фабрика класса

COM-объекты представляют собой экземпляры coclass. Напомним, что Coclass – это класс, поддерживающий один или более интерфейс. COM-объекты могут предоставлять только те услуги, которые определены в интерфейсах coclass. Экземпляры Coclass создаются при помощи специального типа объекта, называемого фабрикой класса.

Фабрика класса – это специальный COM-объект, который поддерживает интерфейс IclassFactory и отвечает за создание экземпляров того класса, с которым ассоциирована данная фабрика класса.

Интерфейс IclassFactory определен в модуле Delphi ActiveX так:

```
type
  IClassFactory = interface (IUnknown)
        ['{00000001-0000-0000-C000-000000000046}']
  function CreateInstance (const unkOuter: lUnknown;
     const iid: TIID out obj): HResult; stdcall;
```

```
  function LockServer (fLock: BOOL): HResult; stdcall;
end;
```

Как видно из вышеприведенного куска кода, интерфейс имеет два метода: `CreateInstance` и `LockServer`. Метод CreateInstance создает экземпляр COM-объекта ассоциированной фабрики класса. Метод LockServer применяется для хранения COM-сервера в памяти. Если параметр метода fLock имеет значение true, то счетчик ссылок сервера увеличивается, иначе – уменьшается. Когда счетчик достигает значения о, сервер выгружается из памяти.

Всякий раз, когда услуги COM-объекта запрашиваются клиентом, фабрика класса создает и регистрирует экземпляр объекта для конкретного пользователя. Если услуга того же COM-объекта запрашивает другой клиент, фабрика класса создает второй экземпляр объекта для обслуживания второго клиента. coclass должен иметь фабрику класса и идентификатор класса CLSID. Использование CLSID для coclass подразумевает, что они могут быть откорректированы всякий раз, когда в класс вводятся новые интерфейсы. Таким образом, в отличие от DLL, новые интерфейсы могут изменять или добавлять методы, не влияя на старые версии.

Мастер создания COM-объектов Delphi самостоятельно заботится о создании фабрики класса.

Локальные и удаленные серверы

С использованием COM клиент не должен беспокоиться о том, где располагается объект, он просто делает вызов интерфейса данного объекта. Технология COM обеспечивает все необходимые шаги для того, чтобы сделать этот вызов. Шаги могут отличаться, в зависимости от местонахождения объекта. Объект может находиться в том же процессе, где и клиент, в другом процессе на том же компьютере, где расположен клиент, или на другом компьютере в сети. В зависимости от этого применяются разные типы серверов:

– внутренний сервер (In-process server);

– локальный сервер или сервер вне процесса (Local server, Out-of-process server);

– удаленный сервер (Remote server).

Внутренний сервер – это библиотека DLL, которая запущена в одном процессе вместе с клиентом. Например, элемент управления ActiveX, который внедрен на Web-страницу и просматривается при помощи Internet Explorer или Netscape Navigator. В данном случае элемент управления ActiveX загружен на клиентскую машину и находится в том же процессе, что и обозреватель Web. Приложение-клиент связывается с сервером внутри процесса при помощи прямых вызовов COM-интерфейса. На рис. 2.3. представлена схема взаимодействия клиента с внутренним сервером.

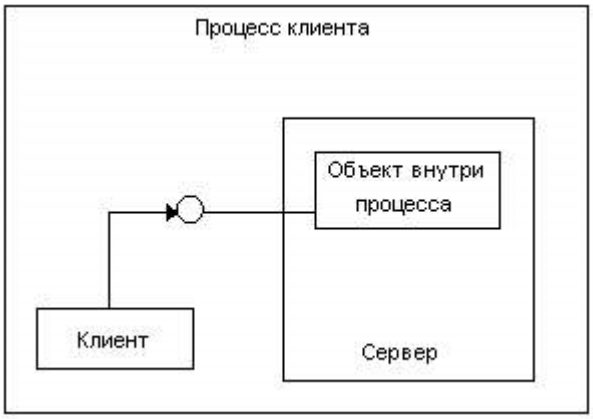


Рис. 2.3. Схема взаимодействия клиента с внутренним сервером

Внутренний COM-сервер должен экспортировать четыре функции:

```
function DllRegisterServer: HResult; stdcall;
function DllUnregisterServer: HResult; stdcall;
function DllGetClassObject (const CLSID, IID: TGUID;
         var Obj): HResult; stdcall;
function DllCanUnloadNow: HResult; stdcall;
```

Все вышеперечисленные функции уже реализованы в модуле comserv, их нужно только добавить в описания exports вашего проекта.

Рассмотрим данные функции более подробно:

– DllRegisterServer – применяется для регистрации DLL COM-сервера в системном реестре Windows. При регистрации COM-класса в системном реестре создается раздел в HKEY_CLASSES_ROOT\CLSID\{XXXXXXXX-XXXX-XXXX-xxxx-xxxxxxxx}, где число, записанное вместо символов x, представляет собой CLSID данного COM-класса. Для внутреннего сервера в данном разделе создается дополнительный подраздел inProcserver32. В этом подразделе указывается путь к DLL внутреннего сервера (рис. 2.4).

– DllUnregisterServer – применяется для удаления всех разделов, подразделов и параметров, которые были созданы в системном реестре функцией DllRegisterServer при регистрации DLL COM-сервера.

– DllGetclassObject – возвращает фабрику класса для конкретного COM-класса.

– DllcanUnloadNow – применяется для определения, можно ли в настоящий момент времени выгрузить DLL COM-сервера из памяти. Функция проверяет, есть ли указатели на любой COM-объект данной DLL, если есть, то возвращает значение S_FALSE, т. е. DLL выгрузить нельзя. Если ни один COM-объект данной DLL не используется, то функция возвращает значение S_TRUE.

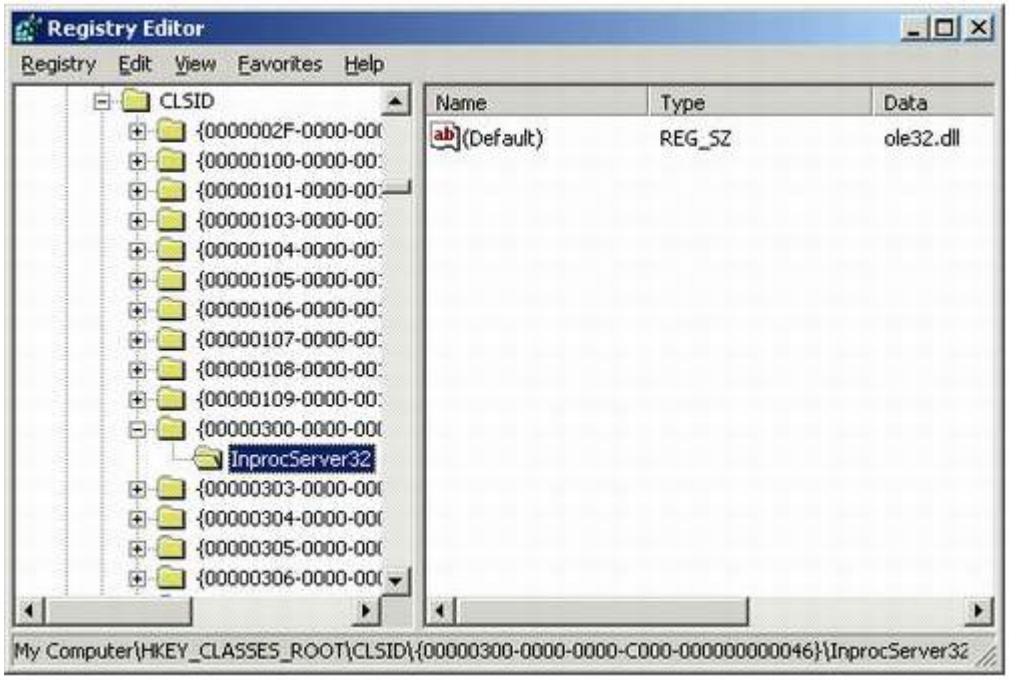


Рис. 2.4. Путь к локальному COM-серверу в окне редактора системного реестра

Локальный сервер – это приложение EXE, которое запущено в другом процессе, но на одном компьютере вместе с клиентом. Например, лист электронной таблицы Microsoft Excel связан с документом Microsoft Word. При этом два разных приложения работают на одном компьютере. Локальные серверы используют COM для соединения с клиентом.

Когда клиент и сервер находятся в различных приложениях, а также, когда они находятся на разных компьютерах в сети, COM использует *внутренний (внутрипроцессный) прокси* (In-process proxy) для реализации процедуры удаленного вызова. Прокси располагается в одном процессе вместе с клиентом, поэтому, с точки зрения клиента, вызов интерфейсов осуществляется так же, как и в случае, когда клиент и сер'вер находятся внутри одного процесса. Задача прокси заключается в том, чтобы перехватывать вызовы клиента и перенаправлять их туда, где запущен сервер. Механизм, который позволяет клиенту получать доступ к объектам, расположенным в другом адресном пространстве или на другом компьютере, называется *маршалинг* (marshaling).

Функции маршалинга:

– принимать указатель интерфейса из процесса сервера и делать указатель прокси в процессе клиента доступным;

– передавать аргументы вызовов интерфейса таким образом, как будто они произошли от клиента и размещать аргументы в процесс удаленного объекта.

Для любого вызова интерфейса клиент помещает аргументы в стек, вызывает необходимую функцию COM-объекта через указатель интерфейса. Если вызов объекта произошел не внутри процесса, вызов проходит через

прокси. Прокси упаковывает аргументы в *пакет маршалинга* и передает получившуюся структуру удаленному объекту. *Заглушка* (stub) объекта распаковывает пакет маршалинга, выбирает аргументы из стека и вызывает необходимую функцию COM-объекта.

Таким образом, маршалинг – это процесс упаковки информации, а демаршалинг – процесс распаковки информации.

Тип маршалинга зависит от объектной принадлежности COM. Объекты могут использовать стандартный механизм маршалинга, предоставляемый интерфейсом IDispatch. *Стандартный маршалинг* позволяет устанавливать связь при помощи стандартного системного *удаленного вызова процедуры* (Remote Procedure Call, RPC).

На рис. 2.5 изображена схема, показывающая методику взаимодействия клиента и сервера в случае, когда приложения работают на одном компьютере, но в разных приложениях.

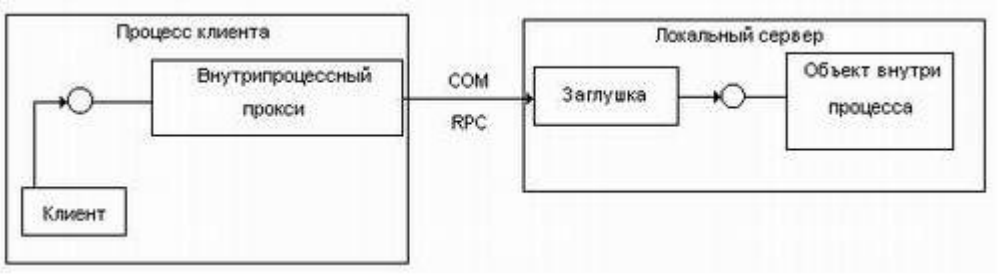


Рис. 2.5. Схема взаимодействия клиента с сервером в разных процессах на одном компьютере

Локальный COM-сервер регистрируется в системном реестре Windows так же, как и внутренний COM-сервер.

Удаленный сервер – это библиотека DLL или иное приложение, запущенное на другом компьютере. То есть клиент и сервер работают на разных компьютерах в сети. Например, приложение базы данных, написанное с помощью Delphi, соединяется с сервером на другом компьютере в сети. Удаленный сервер использует *распределенные COM-интерфейсы* (Distributed COM, DCOM) для связи с клиентом.

Удаленный сервер работает также с помощью прокси. Различие в работе между локальным и удаленным сервером заключается в типе используемой межпроцессной связи. В случае локального сервера – это COM, а в случае удаленного сервера – DCOM. Схема взаимодействия клиента и удаленного сервера показана на рис. 2.6.

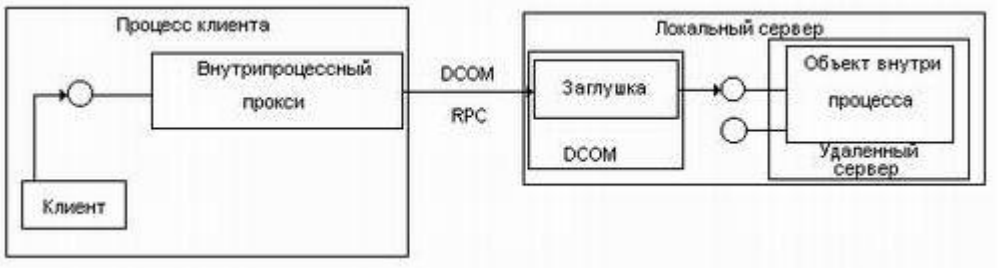


Рис. 2.6. Схема взаимодействия клиента с сервером на разных компьютерах

COM-клиенты

Очень важным при разработке COM-приложений является создание приложений, называемых COM-клиентами, которые могут запрашивать интерфейсы объектов, чтобы определить те услуги, которые может предоставить COM-объект.

Типичным COM-клиентом является диспетчер автоматизации (Automation Controller). *Диспетчер автоматизации* – это часть приложения, которая знает какой тип информации необходим ему от разных объектов сервера, и она запрашивает данную информацию по мере надобности.

Расширения COM

Технология COM изначально разрабатывалась как ядро для осуществления межпрограммного взаимодействия. Уже на этапе разработки предполагалось расширять возможности технологии при помощи так называемых расширений COM. COM расширяет собственную функциональность, благодаря созданию специализированных наборов интерфейсов для решения конкретных задач.

Технология ActiveX – это технология, которая использует компоненты COM, особенно элементы управления. Она была создана для того, чтобы работа с элементами управления была более эффективной. Это особенно необходимо при работе с приложениями Internet/Intranet, в которых элементы управления должны быть загружены на компьютер клиента, прежде чем они будут использоваться.

Технология ActiveX – не единственное расширение COM. В табл. 2.2 представлены некоторые из используемых в настоящее время расширений COM.

Перечисленные в табл. 2.2 расширения COM – это далеко не все из имеющихся. Постоянно идет доработка старых и создание новых, более совершенных технологий межпрограммного взаимодействия.

Таблица 2.2. Список расширений COM

Расширение COM	Краткое описание
Серверы автоматизации (Automation servers)	*Серверы автоматизации*- это объекты, которыми можно управлять из других приложений во время работы приложения. Таким образом, *автоматизация*- это способность приложения программно контролировать объекты других приложений
Диспетчеры автоматизации или COM-клиенты (Automation Controllers, COM Clients)	*Диспетчеры автоматизации*- это клиенты серверов автоматизации. Они позволяют разработчику или пользователю писать сценарии для управления серверами автоматизации
Элементы управления ActiveX (ActiveX Controls)	Элементы управления ActiveX предназначены для серверов внутри процесса (in-process COM servers). Элементы ActiveX обычно используются путем встраивания в приложение-клиент
Библиотеки типов (Type Libraries)	Библиотеки типов представляют собой статичные структуры данных, которые часто сохраняются как файлы ресурсов. Они содержат детализированную информацию об объекте и его интерфейсах. Клиенты серверов автоматизации и элементы управления ActiveX используют данную информацию и всегда считают ее доступной
Страницы активного сервера (Active Server Pages)	*Активные серверные страницы*- это компоненты ActiveX, которые позволяют вам создавать динамически изменяющиеся Web-страницы
Активные документы (Active Documents)	*Активные документы* - это объекты, которые поддерживают связывание и внедрение, визуальное редактирование, перенос (drag-and-drop). В качестве примера таких документов можно представить документы Microsoft Word и книги Microsoft Excel
Визуальные межпроцессные объекты (Visual Cross-process Objects)	*Визуальные межпроцессные объекты*- это визуальные объекты, которыми можно управлять из других процессов

На рис. 2.7 представлена диаграмма, которая показывает связь некоторых расширений COM и их связь с технологией COM.

Использование COM-объектов имеет как преимущества, так и некоторые ограничения. COM-объекты могут быть как визуальными, так и невизуальными. Какие-то COM-объекты должны быть запущены в одном процессе с клиентом, другие – в разных процессах либо на разных,,компьютерах.

Приведенная ниже табл. 2.3 кратко описывает особенности объектов каждого из вышеприведенных расширений COM.

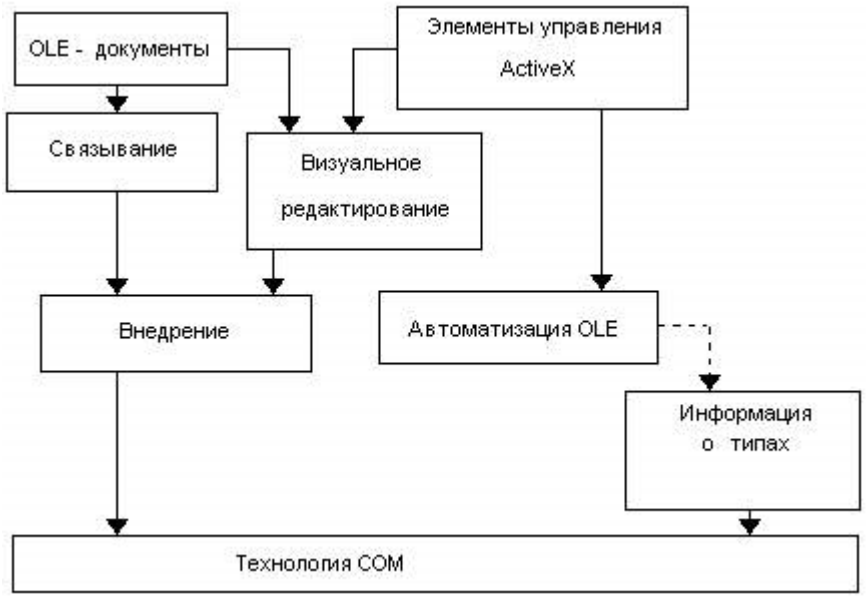


Рис. 2.7. Технологии, основанные на COM

Таблица 2.3. Особенности объектов COM

COM-объект	Визуаль-ность	Процесс	Связь	Библиотека типов
Активный документ (Active Document)	Обычно визуальный	Внутренний или локальный	OLE	Нет
Автоматизация (Automation)	Может быть как визуальным, так и невизуальным	Внутренний, локальный или удаленный	Автоматический маршалинг при помощи интерфейса IDispatch	Требуется для автоматического маршалмнга
Элемент управления ActiveX (ActiveX Control)	Обычно визуальный	Внутренний	Автоматический маршалинг при помощи интерфейса IDispatch	Требуется
Произвольный объект интерфейса	По выбору	Внутренний	Не требуется маршалинг	Рекомендуется
Произвольный объект интерфейса	По выбору	Внутренний, локальный или удаленный	Автоматический маршалинг в зависимости от библиотеки типов, в противном случае-ручной маршалинг	Рекомендуется

3. Сервер автоматизации

Автоматизация – это протокол COM, который определяет, как одно приложение может получить доступ к объектам, находящимся в другом приложении или библиотеке DLL.

Сервер автоматизации – это приложение, которое предоставляет какие-либо услуга приложениям-клиентам. Примерами серверов автоматизации являются такие приложения, как Microsoft Word, Microsoft Excel, Microsoft Internet Explorer и др. Данные приложения могут контролироваться из приложений Delphi или других приложений. Для успешной работы с сервером автоматизации разработчику необходимо знать свойства и методы, которые предоставляют объекты сервера автоматизации. Описания свойств и методов можно получить из руководств разработчика по конкретным приложениям-серверам.

Диспетчер автоматизации (контроллер автоматизации) – это приложение-клиент, которое управляет сервером автоматизации при помощи объектов, которые поддерживают интерфейс IDispatch.

Диспетчеры автоматизации могут быть созданы на любых языках программирования, которые поддерживают технологию COM. Большинство диспетчеров автоматизации написано в настоящее время на таких языках, как C++, Object Pascal (Delphi) и Visual Basic.

В данной главе вы узнаете, как средствами Delphi можно создать диспетчер и сервер автоматизации.

Интерфейс *IDispatch*

Объекты автоматизации представляют собой COM-объекты, которые используют интерфейс IDispatch. Данный интерфейс описан в модуле system следующим образом:

```
type
    IDispatch = interface (IUnknown)
         ['{00020400-0000-0000-COOO-000000000046}']
    function GetTypeInfoCount(out Count: Integer): Integer;
         stdcall;
    function GetTypeInfo(Index, LocaleID: Integer;
         out TypeInfo):Integer; stdcall;
    function GetIDsOfNames(const IID: TGUID; Names: Pointer;
         NameCount, LocaleID: Integer; DispIDs: Pointer):
            Integer; stdcall;
    function Invoke(DispID: Integer; const IID: TGUID;
       LocaleID: Integer; Flags: Word; var Params;
       VarResult, ExceptInfo, ArgErr: Pointer): Integer;
end;
```

Основной функцией интерфейса IDispatch является метод Invoke. Приложение-клиент может вызывать данный метод для выполнения определенных действий на сервере автоматизации. Для того чтобы указать,

какой именно метод хочет вызвать клиент, он передает при вызове метода invoke параметр DispIo. Этот параметр представляет собой число, которое называется *идентификатор диспетчера* (Dispatch ID). Данное число указывает, какой именно метод должен исполняться на сервере. Следующий параметр метода Invoke – параметр IID не используется. Параметр LocaleID содержит информацию о локализации. Параметр Flags указывает, как данный метод будет вызываться: для получения свойств, для установки свойств или обычным способом. Свойство Params содержит указатель на массив TDispParams. Данный массив хранит параметры, передаваемые методу. Параметр VarResult представляет собой указатель на OleVariant, который содержит возвращаемое значение вызываемого метода (если такое имеется), а параметр ExcepInfo – указатель на запись TExcepInfo, которая содержит информацию об ошибке, если метод Invoke возвращает значение DISP_E_EXCEPTION. Последний параметр ArgError – это указатель на целое число, которое является индексом некорректного параметра в массиве Params. В данном случае метод Invoke возвращает значение DISP_E_TYPEMISMATCH или DISP_E_PARAMNOTFOUND.

Следующий метод интерфейса invoke – метод GetIDsOfNames. Он применяется для получения идентификатора диспетчера, одного или нескольких методов по строкам имен этих методов. Параметр IID данного метода не используется. Параметр Names – это указатель на массив имен методов. Такой массив имеет тип PWideChar. Параметр NameCount содержит число, показывающее количество строк в массиве, на который указывает параметр Names. Параметр LocaleID содержит информацию о локализации. Параметр DispIos – это указатель на массив целых чисел NameCount.

Метод GetTypeInfo применяется для получения информации о типе объектов автоматизации. Параметр Index описывает получаемую информацию о типе и должен (обычно) быть равен нулю. Параметр LCID содержит ин формацию о локализации. Если метод выполнился успешно, то параметр TypeInfo будет содержать указатель ITypeInfo на информацию о типе объекта автоматизации.

Метод GetTypeInfoCount используется для получения числа интерфейсов информации о типе, поддерживаемых объектом автоматизации. Число возвращается в параметре Count. Он может содержать два возможных значения: 0, если объект автоматизации не поддерживает информацию о типе, и 1, если поддерживает.

Позднее и раннее связывание

Объекты автоматизации могут работать двумя способами, которые называются *позднее связывание* (late binding) и *раннее связывание* (early binding).

При позднем связывании необходимый метод сервера автоматизации вызывается клиентом внутри метода Invoke интерфейса IDispatch. При использовании позднего связывания вызов метода не разрешен до момента его исполнения, а возможен только с помощью метода Invoke. Во время

компиляции вызов метода сервера автоматизации автоматически преобразуется в вызов метода IDispatch Invoke. Во время выполнения приложения метод Invoke вызывает нужный метод сервера автоматизации.

Раннее связывание означает, что сервер автоматизации предоставляет свои методы при помощи пользовательского интерфейса, который является наследником интерфейса IDispatch. В данном случае диспетчер автоматизации может обращаться к методам сервера автоматизации напрямую, без вызова метода Invoke интерфейса IDispatch. Использование раннего связывания позволяет ускорить работу объектов автоматизации.

Многие объекты автоматизации поддерживают так называемый *двойной интерфейс* (dual interface). Это означает, что такие объекты автоматизации позволяют вызывать методы как из метода Invoke, так и из потомков интерфейса IDispatch. Серверы автоматизации, которые создаются при помощи Delphi, всегда поддерживают двойной интерфейс. Диспетчеры автоматизации, созданные при помощи Delphi, позволяют вызывать методы напрямую внутри интерфейса либо внутри метода Invoke.

Создание диспетчера автоматизации

Одним из самых простых способов создания диспетчера автоматизации является способ импортирования библиотеки типов сервера автоматизации. При этом вы можете использовать автоматически генерируемые классы для управления сервером автоматизации.

Для того чтобы импортировать библиотеку типов, нужно выполнить следующее:

1. Выбрать в главном меню Delphi пункт **Project / Import Type Library** (Проект / Импорт библиотеки типов).

2. В появившемся диалоговом окне (рис. 3.1) выбрать нужную библиотеку типов из представленного списка.

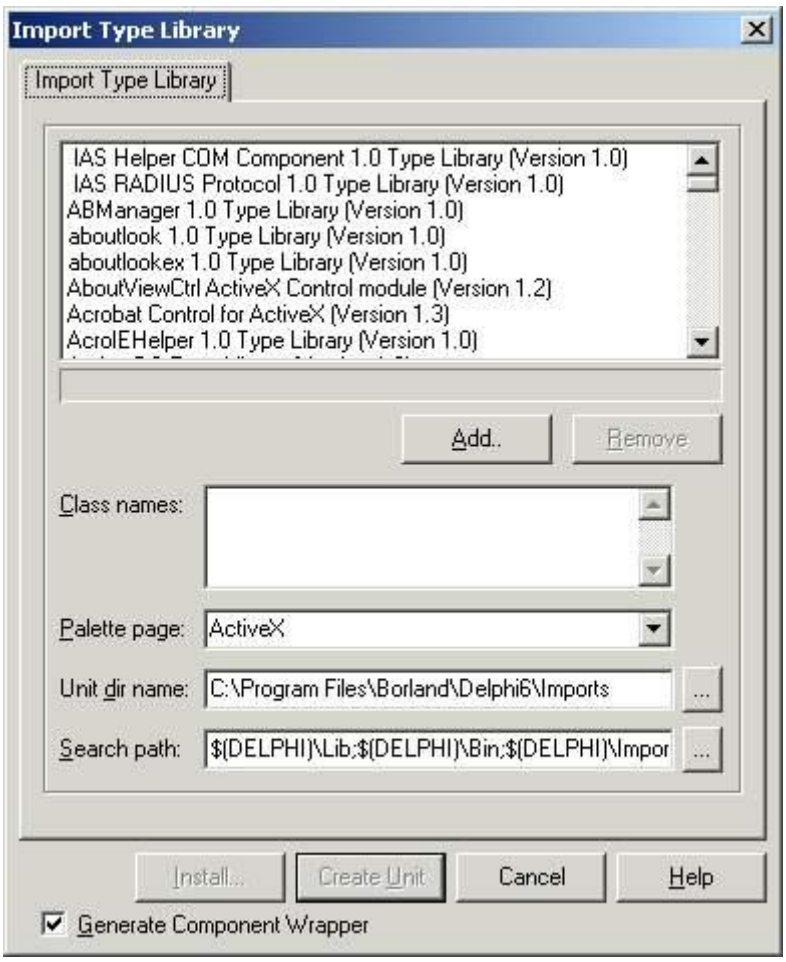


Рис. 3.1. Диалоговое окно импортирования библиотеки типов

3. В выпадающем списке **Palette page** (Страница палитры компонентов) диалогового окна выберите страницу палитры компонентов Delphi, на которую будет размещен выбранный вами сервер автоматизации.

4. Убедитесь в том, что флажок **Generate Component Wrapper** (Создать суперобложку компонента) включен.

5. Нажмите **кнопку Install** (Установить).

Обработка событий диспетчера автоматизации

Как только вы инсталлировали необходимый сервер автоматизации в палитру компонентов, вы можете использовать инспектор объектов для написания обработчиков событий. Обработка событий осуществляется так же, как и обработка событий других компонентов Delphi:

1. Поместите на форму сервер автоматизации из палитры компонентов.

2. Щелкните на компоненте, затем нажмите вкладку **Events** в окне инспектора объектов – вы увидите список событий данного компонента.

3. Дважды щелкните напротив нужного события, после чего Delphi сгенерирует заготовку для обработчика события, в который вы можете поместить свой код.

После того как вы напишете необходимые обработчики событий, вы можете приступать к подключению к серверу автоматизации.

Подключение к серверу автоматизации

Рассмотрим процесс подключения к серверу автоматизации на примере таких программ, как Microsoft Word, Microsoft Excel и Microsoft Outlook, входящих в состав популярного программного пакета Microsoft Office. Подключение к другим серверам автоматизации может немного отличаться, но принцип работы везде одинаков.

Прежде чем переходить непосредственно к подключению к серверу автоматизации, рассмотрим объектную модель Microsoft Office.

В Microsoft Office нет понятия наследование, вместо него используется так называемое *встраивание.* Встраивание помогает получать новые классы. Итак, в любом приложении Microsoft Office всегда имеется центральный *базовый* объект. Для Microsoft Word это Word.Application, для Microsoft Excel – Excel.Application. Microsoft Outlook сам является базовым объектом и называется OutlookApplication, однако, несмотря на это, в объект Application встраиваются все остальные объекты, которые, в свою очередь, являются свойствами базового объекта.

Различные объекты приложений Microsoft Office имеют самые разнообразные методы, но некоторые из них одинаковы для разных приложений. К числу совпадающих методов относятся Run, Activate и др.

Примечание

Разница в этих методах все же есть. В различных приложениях Microsoft Office они имеют разные параметры. Действия методов тоже могут отличаться.

Рис. 3.2 показывает небольшую часть структуры объекта Microsoft Word Object. Более полную информацию об объектной архитектуре Microsoft Office можно найти в справочных материалах по Microsoft Office.

Рис. 3.2. Фрагмент структуры объекта Microsoft Word Object

Как только открывается новый документ, приложения Microsoft Office, автоматически создается каркас нового документа, который представляет собой набор библиотек с классами. Объекты этих классов будут доступны в данном документе. Задачей разработчика диспетчера автоматизации является получить доступ к корневому объекту сервера, выстроить цепочку доступа к встроенным объектам и правильно передать параметры.

Итак, базовым объектом любого приложения Microsoft Office является объект Application. Попробуем получить к нему доступ средствами Delphi.

Итак, выполним следующие шаги:

1. Создадим новый проект Delphi с помощью пункта главного меню **File/New Application** (Файл/Новое приложение).

2. Поместим на форму компонент WordApplication с вкладки **Servers** палитры компонентов Delphi.

3. Установим свойства AutoConnect и AutoQuit компонента WordApplication в значение true (эти свойства отвечают за автоматическую загрузку и выгрузку из памяти сервера автоматизации после запуска приложения).

4. Запустим приложение с помощью пункта главного меню **Run/Run** (Запуск/Запуск).

В результате, хотя визуально ничего не наблюдается, кроме отображения формы, приложение проделало большую работу. Наше приложение запустило сервер автоматизации Microsoft Word. Чтобы убедиться в этом, достаточно посмотреть список задач, выполняемых Windows во время работы нашего приложения. Нажмите комбинацию клавиш <Ctrl>+<Alt>+<Del> во время работы приложения и убедитесь, что среди задач, выполняемых Windows, появилась задача **Winword.exe.**

В общих чертах, наше приложение выполнило следующее:

- при запуске приложения, в системном реестре Windows был найден сервер автоматизации WordApplication при помощи идентификатора (CLSID);
- запустилось приложение, находящееся по адресу, указанному в системном реестре Windows;
- сервер автоматизации предоставил нашему приложению (простейшему диспетчеру автоматизации) интерфейс, через который приложение может получить доступ к базовому объекту Microsoft Word.

Теперь закройте приложение и посмотрите еще раз на задачи, выполняемые Windows. Сервер автоматизации выгружен из памяти компьютера при помощи интерфейса IDispatch (данный интерфейс имеет метод AddRef, при помощи которого можно определить число клиентов, которые в настоящий момент пользуются услугами сервера).

Тот же результат можно получить, если использовать код, приведенный на листинге 3.1. Для этого разместите на форме две кнопки **Start** и **Finish.**

Листинг 3.1

```
unit Unit1;

interface

uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls,
   Forms, Dialogs, StdCtrls, COMObj, OleServer, WordXP;
```

```
type
   TForm1 = class(TForm)
      Start: TButton;
      Finish: TButton;
      procedure StartClickfSender: TObject);
      procedure FinishClick(Sender: TObject);
   end;

var
   Form1: TForm1;
   wd: OleVariant;
   fileName: string;

implementation

 {$R *.DFM}

procedure TForm1.StartClick(Sender: TObject);
begin
   try
      fileName:= ExtractFilePath(Application.EXEName) +
         'report.DOC';
      { Создаем объект интерфейса для доступа к серверу COM }
      wd:= CreateOleObject('Word.Application');
      { Проверка наличия методов и правильность передачи
        параметров будет осуществляться на стадии выполнения
        приложения }
      wd.Application.Documents.Add;
      wd.Application.Activedocument.Range.Insertafter(Now);
      wd.Application.Activedocument.SaveAs(Filename);
   except
   end;
end;

procedure TForm1.FinishClick(Sender: TObject);
begin
   // Выгружаем сервер из памяти компьютера
   wd.Application.Quit(true,0);
end;

end.
```

Обратите внимание на необходимость добавления модуля ComObj в раздел Uses. Как вы можете видеть, вкладка **Servers** палитры компонентов содержит много компонентов. Кроме базовых объектов серверов автоматизации, данная вкладка содержит несколько вложенных объектов, таких как документ Microsoft Word (WordDocument), рабочая Книга Excel (ExcelWorkBook) и др.

Отметим, что все компоненты, содержащиеся на вкладке **Servers,** являются наследниками класса TOLEServer, который, в свою очередь, происходит от класса TComponent. Класс TOLEServer – базовый класс для всех COM-серверов, которые можно получить путем импортирования библиотек типов серверов автоматизации. Данный класс имеет несколько свойств и методов, позволяющих управлять связью с COM-сервером. Например, свойство AutoConnect, которое, в случае, если оно имеет значение true, автоматически запускает COM-сервер и производит извлечение интерфейса для связи сервера и диспетчера автоматизации.

Рассмотрим очень важное свойство данного класса ConnectKind (тип подключаемого процесса). Это свойство используется методом Connect, который вызывается автоматически при AutoConnect = true. В табл. 3.1 представлены значения свойства ConnectKind.

Таблица 3.1. Значения, принимаемые свойством ConnectKind

Значения свойства ConnectKind CkRunningOrNew	Описание Диспетчер автоматизации подключается к уже существующему процессу. В случае, если процесс не запущен - запускает и подключается к нему. Этот вид взаимодействия между диспетчером и сервером автоматизации является наиболее часто применяемым. Данное значение свойства устанавливается по умолчанию
CkNewInstance	Диспетчер автоматизации в любом случае создает новый экземпляр сервера автоматизации
CkRunningInstance	Соединение устанавливается только с уже запущенные сервером автоматизации, если такой, не обнаружен - генерируется ошибка
CkRemote	Применяется для установления соединения с сервером автоматизации, который располагается на удаленном компьютере в сети. При использовании данного значения свойства ConnectKind необходимо указать имя удаленного компьютера в свойстве RemoteMachineName
CkAttachToInterface	В данном случае соединение не создается, поэтому нельзя устанавливать значение true для свойства AutoConnect. Соединение с сервером в этом случае осуществляется при помощи метода ConnectTo

Примечание

При помощи последнего значения свойства ConnectKind – CkAttachToInterface – можно последовательно подключать к уже

существующему интерфейсу несколько компонентов, таких как WordDocument или WordFont. Установка данного значения свойства необходима, когда диспетчер автоматизации должен отслеживать события, которые происходят в сервере автоматизации.

Свойство ConnectKind можно устанавливать в окне инспектора объектов при помощи выпадающего списка, который появляется при щелчке мышью на данном свойстве.

Управление сервером автоматизации

Рассмотрим в качестве примера применения свойства ConnectKind подключение и управление сервером автоматизации. Выполните последовательно следующие шаги:

1. При помощи пункта главного меню **File/New Application** (Файл/Новое приложение) создайте новый проект.

2. Поместите на форму компоненты WordApplication и WordDocument, расположенные на вкладке **Servers.**

3. Установите свойства AutoConnect и AutoQuit для компонента WordApplication в true.

4. Установите свойство ConnectKind для компонента WordDocument в CkAttachToInterface.

5. Выберите на форме компонент WordApplication и в окне инспектора объектов перейдите на вкладку **Events** (События).

6. Дважды щелкните на событии onDocumentChange и запишите в заготовке обработчика события, которую создаст Delphi, приведенный на листинге 3.2 код

Листинг 3.2

```
procedure TForm1.WordApplication1DocumentChange(Sender:
     TObject);
begin
// Производим подключение к активному документу Microsoft Word
   WordDocument1.ConnectTo(WordApplication1.ActiveDocument);
   { Наш диспетчер автоматизации добавляет новую строку
     в текущий документ }
   WordDocument1.Range.InsertAfter(#13 +
      'Переход к документу ' + #13 +
      WordApplication1.ActiveDocument.FullName +
      ' произведен: '+ DateTimeToStr(Now));
end;
```

Рассмотрим, какие действия произведет код, представленный на листинге 3.2. Во-первых, данный код будет выполняться всякий раз при смене текущего документа Microsoft Word. После смены текущего документа (то есть, простого переключения между несколькими документами Microsoft Word) наш диспетчер автоматизации при помощи метода ConnectTo подключится к активному документу. Затем, при помощи метода InsertAfter

производится вставка текстовой строки в текущий документ. Переход на новую строку в документе осуществляется при помощи вставки символа перевода строки (#13). Метод FullName позволяет получить название текущего документа Microsoft Word.

7. Для события формы **Create** напишите код, представленный на листинге 3.3.

Листинг 3.3

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   // Отображение сервера автоматизации на экране
   WordApplication1.Visible:=true;
end;
```

Данный код отобразит Microsoft Word на экране, даже если он ранее не был загружен.

Запустите созданный вами диспетчер автоматизации при помощи пункта главного меню **Run/Run** (Запуск/Запуск). После старта приложения будет автоматически загружен Microsoft Word. С помощью пункта меню Microsoft Word **File/New** (Файл/Создать) создайте несколько новых документов. Теперь убедимся, что наш диспетчер автоматизации работает. Попробуйте переключаться между документами при помощи раздела меню Microsoft Word **Window** (Окно). Вы можете видеть, как диспетчер автоматизации добавляет новые строки в текущий документ.

Такой же принцип управления можно применять и в случае, когда необходимо контролировать действия Microsoft Excel. Когда в Microsoft Excel создается новая книга, возникает событие OnNewWorkBook.

Создание сервера автоматизации

Сервером автоматизации может являться либо приложение, либо DLL. Рассмотрим шаги, которые вам предстоит выполнить для создания сервера автоматизации.

1. Создайте приложение или DLL, которое должно выступать в роли сервера автоматизации. Можно использовать любое ранее созданное вами приложение и добавить к нему автоматизацию.

2. При помощи мастера объекта автоматизации создайте объект автоматизации и добавьте его к проекту.

3. Добавьте свойства и методы к вашему объекту автоматизации при помощи библиотеки типов. Данные свойства и методы нужны для того, чтобы диспетчеры автоматизации могли их использовать при обращении к вашему серверу автоматизации.

4. Зарегистрируйте приложение как сервер автоматизации. Теперь рассмотрим эти шаги более подробно.

Создадим самое простое приложение, состоящее из одной только формы. Для этого достаточно лишь запустить Delphi и выбрать пункт меню **File/New Application** (Файл/Новое приложение).

Теперь добавим к нашему проекту объект автоматизации для того, чтобы приложение выполняло функции сервера автоматизации. Чтобы сделать это, нужно выбрать пункт главного меню **File/New** (Файл/Новый). После чего перейдите в открывшемся окне **New Items** (Новые объекты) на вкладку **ActiveX** (рис. 3.3).

Выберите пиктограмму **Automation Object** (Объект автоматизации) двойным щелчком. После этого появится окно **Automation Object Wizard** (Мастер объекта автоматизации) (рис. 3.4).

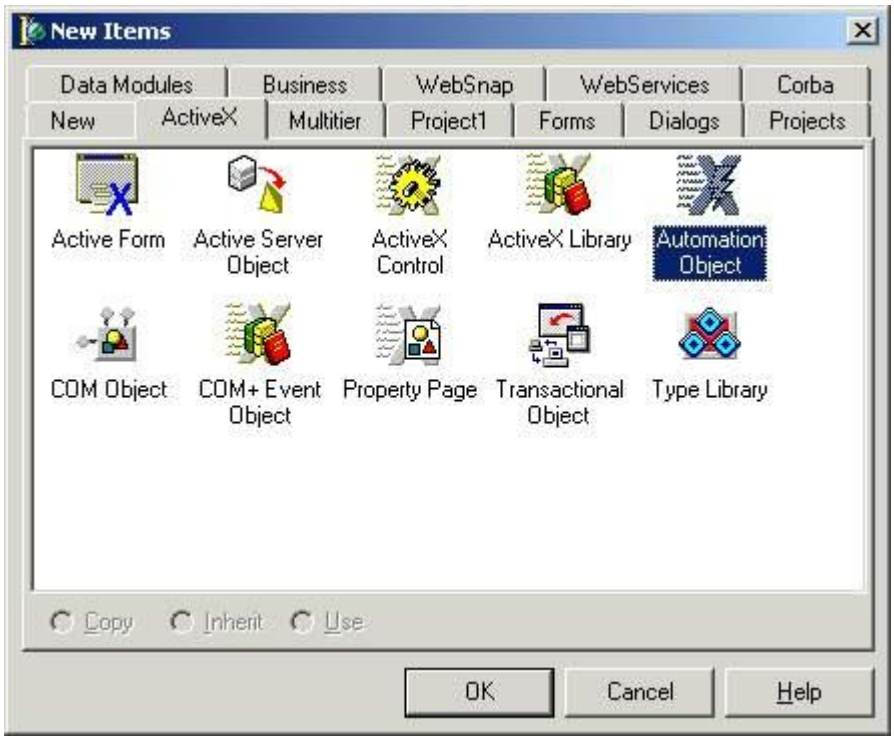


Рис. 3.3. Вкладка ActiveX окна **New Items**

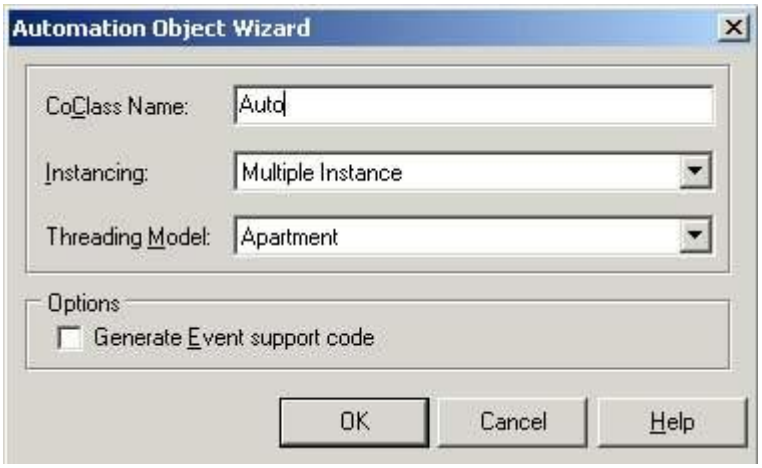


Рис. 3.4. Диалоговое окно **Automation Object Wizard**

Впишем имя Auto в поле **CoClass Name** для COM-класса объекта автоматизации. Мастер автоматически добавит букву T к имени класса, если

создается класс Object Pascal для объекта автоматизации, или букву I – при создании основного интерфейса для объекта автоматизации.

Возможные значения остальных полей ввода нами уже рассматривались в предыдущей главе.

После установки всех параметров в данном диалоговом окне Delphi создает новую библиотеку типа для проекта, и, кроме того, добавит интерфейс и класс компонентов к данной библиотеке типа. Более того, мастер создаст новый модуль в нашем проекте, который содержит реализацию интерфейса автоматизации, добавленного к библиотеке типа. На рис. 3.5 изображен редактор библиотеки типа сразу же после закрытия диалогового окна Automation Object Wizard (Мастер объекта автоматизации).

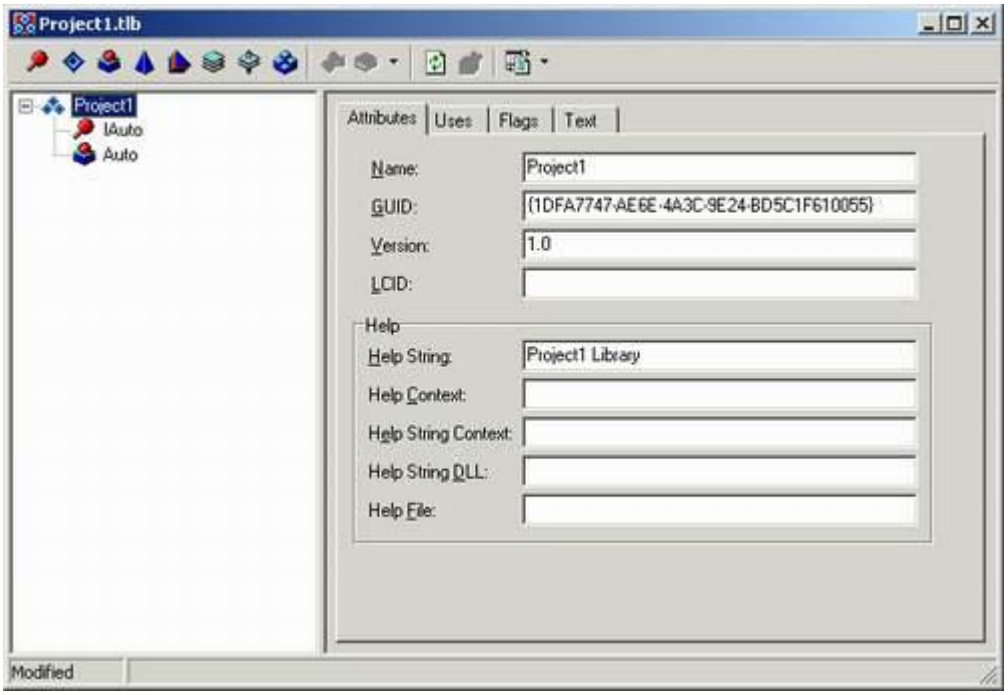


Рис. 3.5. Новый проект автоматизации в редакторе библиотеки типа

В листинге 3.4 приведен код нового модуля, который генерирует мастер объекта автоматизации Delphi.

Листинг 3.4.

```
unit Unit2;

interface

uses
   ComObj, ActiveX, Project1_TLB, StdVcl;

type
   TAuto = class(TAutoObject, IAuto)
end;

implementation

uses ComServ;
```

```
initialization

TAutoObjectFactory.Create(ComServer, TAuto,   Class_Auto,
   ciMultiInstance, tmApartment);
end.
```

Из вышеприведенного листинга ясно, что новый объект автоматизации TAuto является классом-наследником класса TAutoObject. Отметим, что класс TAutoObject является базовым классом всех объектов автоматизации.

Теперь, после добавления к нашему проекту нового объекта автоматизации TAuto мы можем дописать к основному интерфейсу IAuto необходимые свойства и методы.

Поставим перед собой простую задачу. Допустим, наш сервер автоматизации должен изменять цвет формы путем передачи необходимого цвета из диспетчера автоматизации. Для решения данной задачи добавим свойство Color к интерфейсу IAuto. Это можно сделать в редакторе библиотеки типов разными способами, один из которых такой:

- щелкните левой кнопкой мыши на изображение интерфейса IAuto в левой части редактора библиотеки типа;
- т. к. нам нужно только получать цвет от диспетчера автоматизации, то нам понадобится свойство только для чтения, для этого щелкните на стрелочке справа от пиктограммы, изображающей свойство в верхней панели редактора, и выберите **Write Only** (Только для записи) (рис. 3.6);
- назовите свойство Color и установите тип свойства OLE_COLOR в поле Type.

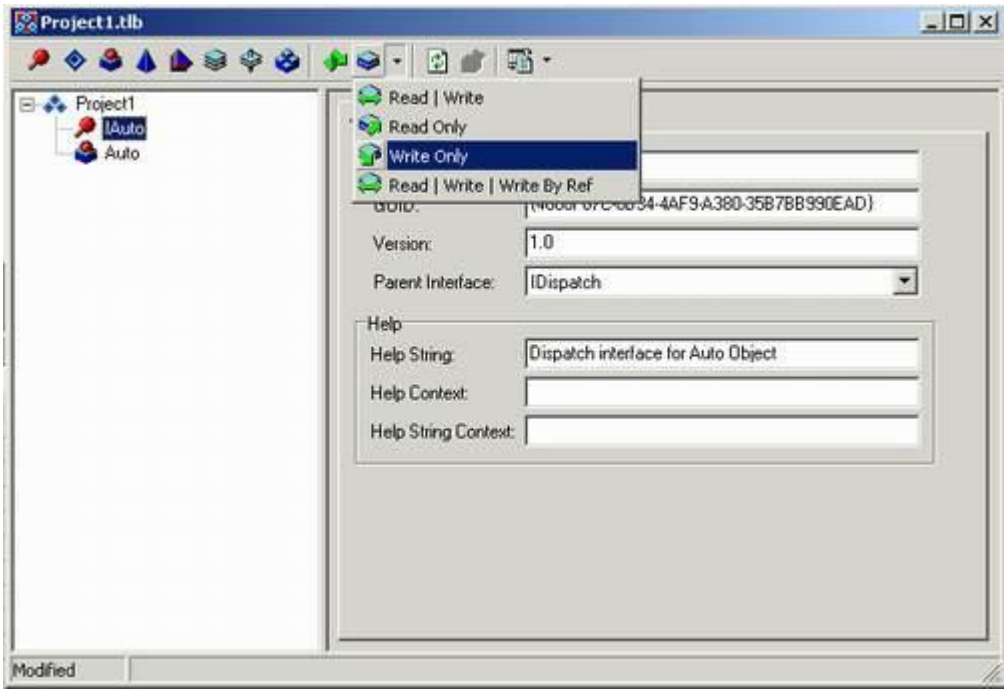


Рис. 3.6. Добавление свойства только для чтения в интерфейс IAuto

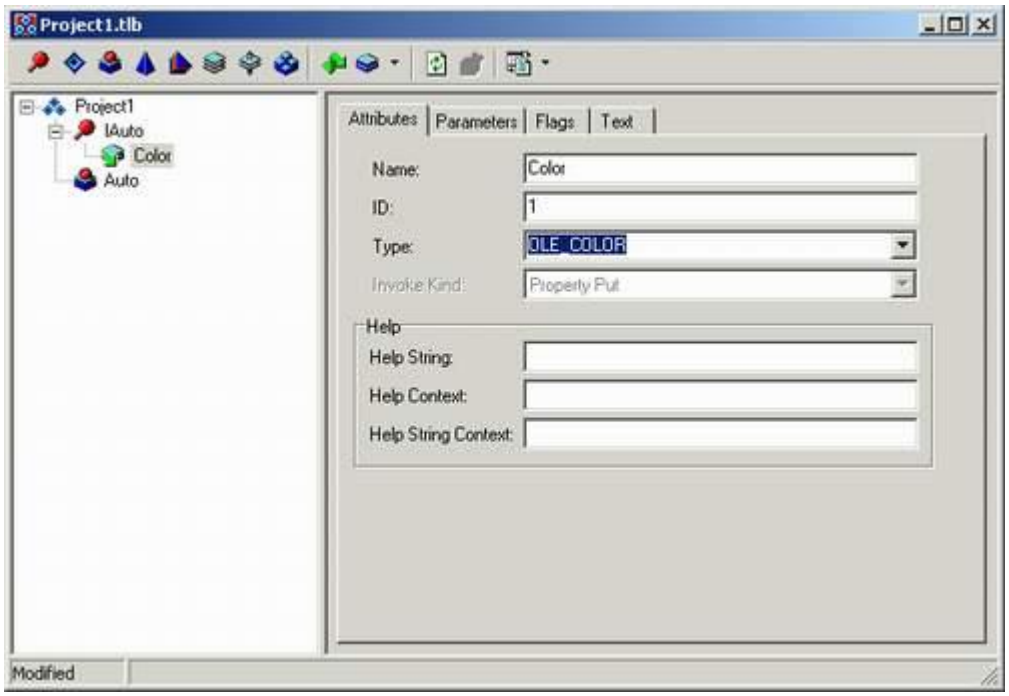


Рис. 3.7. Законченная библиотека типов

Все. Новое свойство добавлено, теперь нажмите в верхней панели редактора кнопку **Refresh Implementation** для обновления модуля реализации объекта автоматизации (рис. 3.7). В результате Delphi подкорректирует модуль реализации в соответствии с теми изменениями, которые были внесены в редакторе библиотеки. Код измененного модуля представлен на листинге 3.5. Кроме того, в данном листинге вручную дописано тело процедуры TAuto.Set_Color.

Листинг 3.5

```
unit Unit2;

interface

uses
   ComObj, ActiveX, Test_TLB, StdVcl;

type
   TAuto = class(TAutoObject, IAuto)
   protected
      procedure Set_Color(Value: OLE_COLOR); safecall;
end;

implementation

uses ComServ, Unit1;

procedure TAuto.Set_Color(Value: OLE_COLOR);
begin
  Form1.Color:=Value;
end;

initialization
```

```
   TAutoObjectFactory.Create(ComServer, TAuto, Class_Auto,
      ciMultiInstance, tmApartment);
end.
```

После этого создадим на жестком диске новую папку. Назовем ее, например, Automation. Сохраним наше приложение в эту папку. При сохранении назовём проект Test. Затем надо откомпилировать (или просто запустить) приложение для того, чтобы получить исполняемый файл Test.exe. В этом процессе параллельно выполняется регистрация нашего интерфейса IAuto в системном реестре, поэтому вполне возможно, что нам понадобятся права администратора, причём эти права нужны при запуске Delphi, ведь это именно Delphi будет добавлять запись в реестр. То есть, если при компиляции будет выдана ошибка на эту тему, то значит, надо перезагрузить Delphi с правами администратора и повторить компиляцию.

В результате наш интерфейс IAuto будет добавлен в системный реестр:

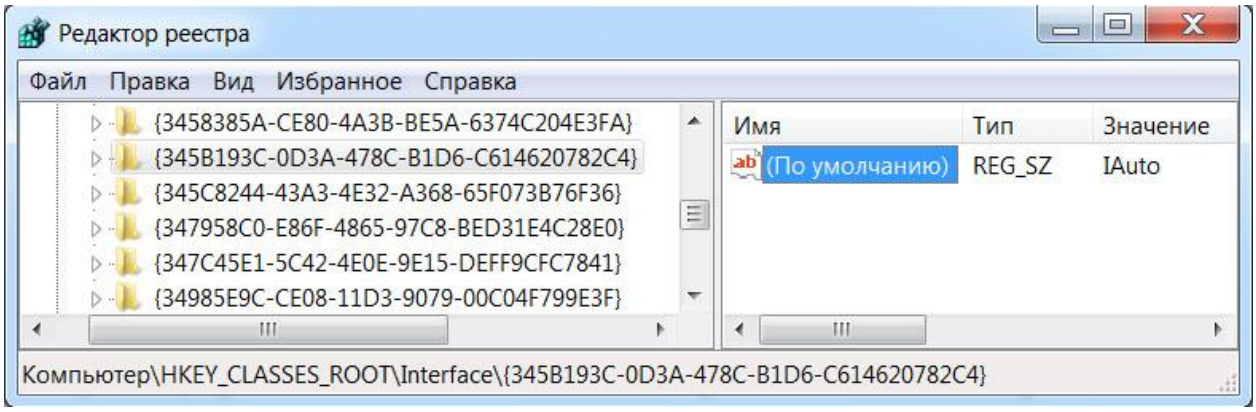


Рис. 3.8. Интерфейс IAuto в системном реестре

Итак, наш новый сервер автоматизации готов. Для того чтобы убедиться, что он работает, создадим диспетчер автоматизации. Для этого закроем текущий проект и создадим новый **(File / New Application).** Разместим на форме две кнопки: **Соединение** и **Изменить цвет** (рис. 3.9).

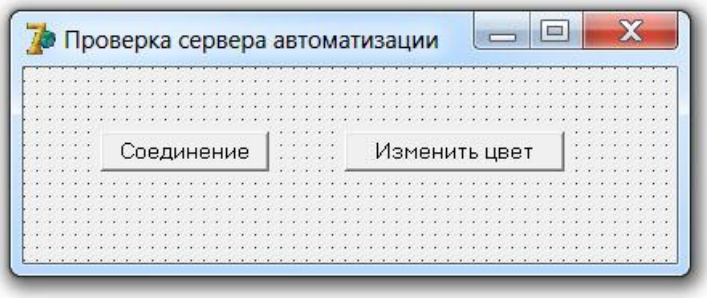


Рис. 3.9. Внешний вид формы диспетчера автоматизации

Приведенный листинг 3.6 иллюстрирует код для нашего диспетчера автоматизации.

Листинг 3.6

```
unit Unit1;

interface
```

```
uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls,
   Forms, Dialogs, StdCtrls, Test_TLB;

type
   TForm1 = class(TForm)
   Button1: TButton;
   Button2: TButton;
   procedure Button1Click(Sender: TObject);
   procedure Button2Click(Sender: TObject);
private
   FIntf: IAuto;
end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
   FIntf:= CoAuto.Create;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   FIntf.Set_Color(clRed);
end;

end.
```

Рассмотрим вышеприведенный листинг. Итак, для начала нужно добавить в раздел uses библиотеку типов Test_TLB. Далее в разделе private добавим поле FIntf типа IAuto. После чего для кнопки **Соединение** пишем обработчик события OnClick, которое создает экземпляр сервера автоматизации. При нажатии на эту кнопку будет запущено приложение Test.exe. Для события OnClick кнопки **Изменить цвет** пишем обработчик события, который вызывает метод Set_Color интерфейса IAuto. Установим, например, красный цвет clRed формы приложения-сервера. Запустим наш диспетчер автоматизации на исполнение. Нажмем последовательно кнопки **Соединение** и **Изменить цвет.** Вы можете видеть, как сначала запускается приложение Test.exe (если оно не было ещё запущено), а затем цвет формы приложения Test.exe изменяется на красный.

Наше простейшее приложение, использующее автоматизацию успешно работает (рис. 3.10).

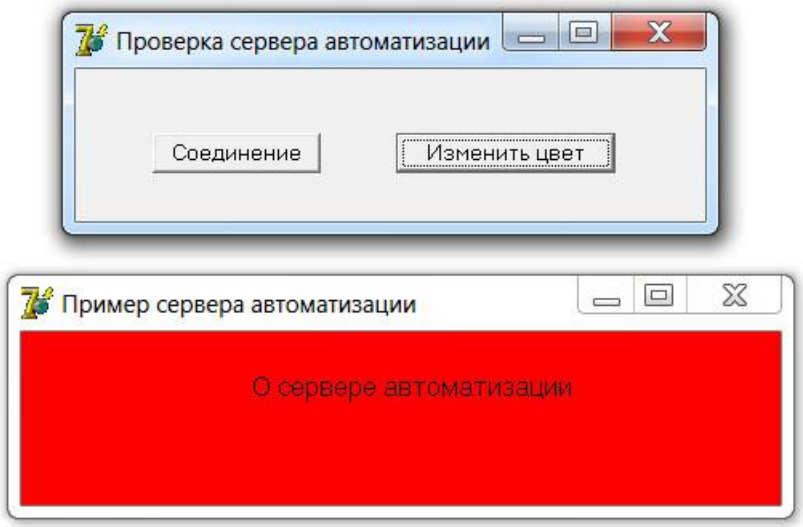


Рис. 3.10. Результат работы приложения автоматизации

4. Технология ActiveX

Эта глава посвящена обсуждению использования готовых элементов ActiveX, предоставляемых другими разработчиками. Мы рассмотрим, что такое элемент управления ActiveX, в каких случаях его необходимо применять. Научимся вносить элемент управления ActiveX в палитру компонентов.

Что такое элемент управления ActiveX?

Ответ на вопрос, заданный в заголовке данного раздела, может быть разным. Все зависит от того, с какой точки зрения смотреть на ActiveX. С точки зрения разработчика, элемент управления ActiveX – это нечто, обладающее свойствами, событиями и методами (практически как любой другой компонент). Разработчик на Delphi может не иметь представления о том, что такое COM, и, в то же время, успешно использовать элементы управления ActiveX в своих приложениях.

С точки зрения компонентной модели объектов (COM), элемент управления ActiveX представляет собой сервер автоматизации, который реализован в виде DLL и исполняемый в одном процессе с вашим приложением. Элементы управления ActiveX допускают визуальное редактирование, т. е. вы можете изменять значения их свойств, методов, писать обработчики событий точно так же, как вы делали это для компонентов из VCL Delphi. Вообще, следует заметить, что идея технологии ActiveX была частично реализована достаточно давно. Еще в Microsoft Visual Basic для разработки 16-разрядных приложений были использованы так называемые модули расширения VBX. Разработчики быстро поняли все преимущества данной технологии и создали тысячи модулей VBX. Идея компонентной разработки понравилась многим, и по ее принципу стали создавать многие средства для разработки приложений, к числу которых относится и Delphi.

Данная технология постепенно переросла в 32-разрядную технологию ActiveX.

Необходимость использования ActiveX

Вам могло и не приходить в голову, что вы уже использовали элементы управления ActiveX при написании своих приложений. Дело в том, что многие элементы управления ActiveX, которые зарегистрированы в системном реестре Windows, уже установлены в палитру компонентов Delphi и внешне ничем не отличаются от обычных компонентов Delphi. Мы не будем останавливаться на том, какие компоненты являются элементами управления. Вместо этого посмотрим, когда возникает необходимость использовать ActiveX.

Обычно такая необходимость возникает, когда вы хотите расширить функциональные возможности своего приложения за счет возможностей уже зарегистрированных в системе приложений. Например, если вы хотите создать собственный обозреватель Web, вам необязательно начинать писать

его с нуля. Есть ведь великолепное ядро для обозревателя Web, которое используется в приложении Microsoft Internet Explorer (SHDOCVW.DLL). Вашей задачей здесь является лишь подключение данного ядра (которое является элементом управления ActiveX) к своему приложению и работа с ним.

Отметим, что в пятой версии Delphi в палитру компонентов была добавлена вкладка **Internet,** которая предоставляет возможность разработчику создавать приложения на основе ядра Microsoft Internet Explorer.

Внесение элемента управления ActiveX в палитру компонентов

Вместе с Delphi поставляется несколько компонентов ActiveX, которые были сделаны различными разработчиками, но, скорее всего, вам понадобится самим добавлять новые элементы ActiveX в палитру компонентов Delphi.

Установка нужного элемента управления ActiveX начинается с выбора пункта главного меню **Component / Import ActiveX Control** (Компонент / Импорт элемента управления ActiveX). Появится диалоговое окно (рис. 4.1), содержащее сведения обо всех элементах управления ActiveX, которые были зарегистрированы в системном реестре Windows.

Рассмотрим данное диалоговое окно.

В верхней части окна перечислены зарегистрированные в системе элементы ActiveX. Список **Class names** (Имена классов) отображает названия классов, имеющихся в данном элементе управления. На рис. 4.1 мы видим, что у выбранного элемента ActiveX имеются два встроенных класса TDHTMLEdit и TDHTMLSafe.

Выпадающий список **Palette page** (Вкладка палитры) служит для выбора вкладки палитры компонентов, на которую будет размещен выбранный элемент ActiveX. По умолчанию, это вкладка **ActiveX.**

Следующие два поля для ввода **Unit dir name** (Имя директории для модуля) и **Search path** (Путь для поиска) предназначены, соответственно, для указания директории, в которой будет размещен модуль элемента ActiveX, и путь поиска данного модуля для компилятора Delphi.

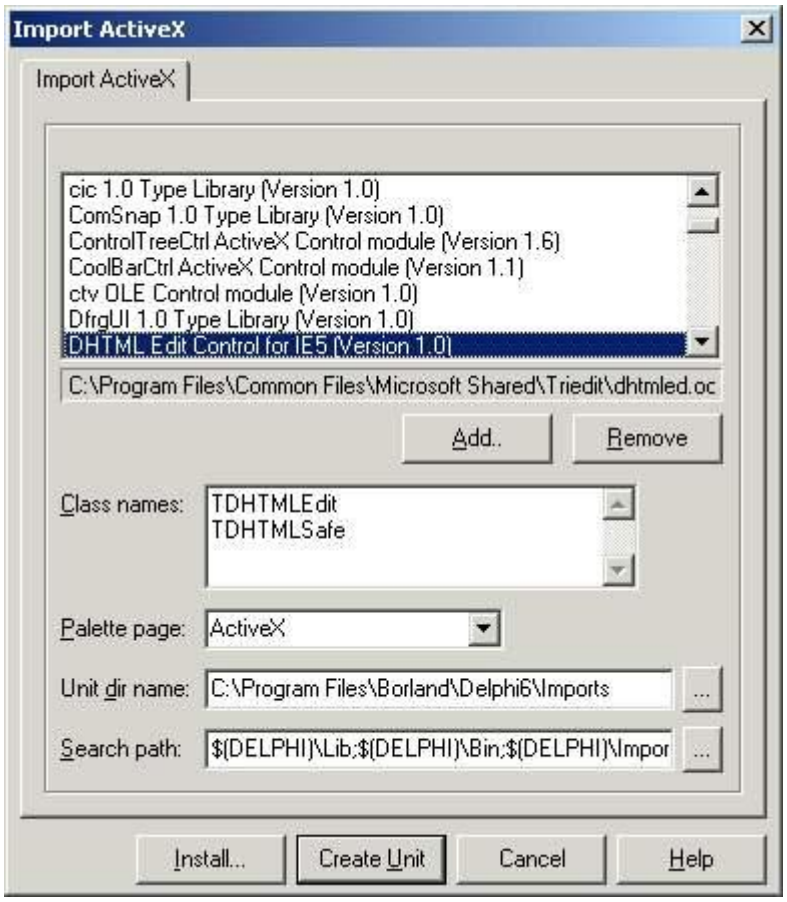


Рис. 4.1. Диалоговое окно импорта элементов управления ActiveX

Выберите тот же элемент управления ActiveX, какой изображен на рис. 4.1. Если такого элемента у вас нет, то возьмите любой другой, но в дальнейшем мы будем описывать установку именно этого элемента управления.

После того как вы выбрали данный элемент управления, нажмите кнопку **Install** (Установить). Появится окно **Install** (Установка) (рис. 4.2).

Примечание

Другая кнопка, ***Create Unit*** *(Создать модуль), позволяет создать так называемый файл представления (wrapper). Данный файл – это описание библиотеки типов, он содержит описание всех методов, событий и свойств, которые находятся в элементе управления. Содержимое данного файла написано на языке Object Pascal. Имя файла состоит из двух частей: имени элемента ActiveX и строки _TLB.PAS. Нажав кнопку* ***Create Unit*** *(Создать модуль) вы лишь создадите данный файл, после чего можете посмотреть его содержимое. Для продолжения установки элемента управления вам нужно воспользоваться кнопкой* ***Install*** *(Установка). Данное окно содержит две вкладки* ***Into existing package*** *(В существующий пакет) и* ***Into new package*** *(В новый пакет). Вы должны выбрать, в какой пакет хотите включить новый элемент управления.*

Примечание

Рекомендуется для элементов управления ActiveX создать свой собственный пакет. Это позволит экономнее использовать ресурсы. Мы в целях упрощения описания установки не будем создавать новый пакет.

Добавим наш элемент управления в пакет, предлагаемый Delphi по умолчанию (dclusrSO.dpk).

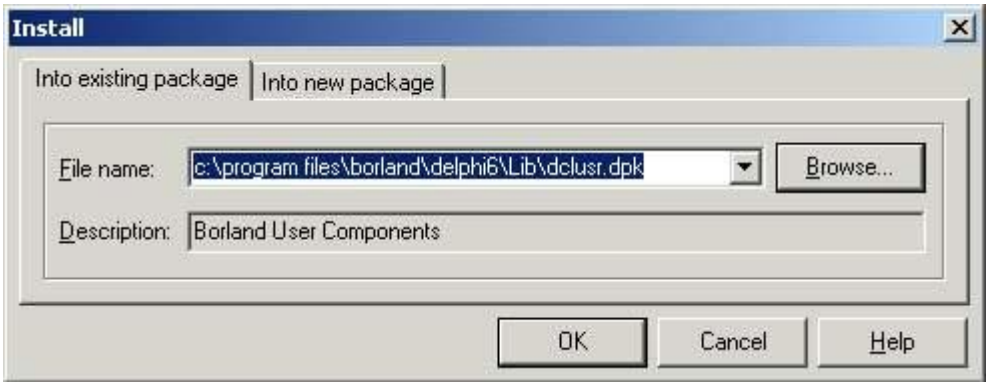


Рис. 4.2. Окно установки элемента управления в пакет

После нажатия кнопки OK в окне **Install** (Установка) Delphi откомпилирует пакет dclusrSO.dpk. В результате, содержимое пакета должно выглядеть, как представлено на рис. 4.3.

Рис. 4.3. Содержимое пакета dclusr5O.dpk после установки нового элемента управления ActiveX

Теперь перейдем к вкладке **ActiveX** палитры компонентов и посмотрим, какие изменения там произошли (рис. 4.4).

Рис. 4.4. Содержимое вкладки **ActiveX**

Как вы можете видеть, на вкладке появились пиктограммы двух новых компонентов: DHTMLEdit и DHTMLSafe.

Теперь вы можете размещать любой из этих компонентов на ваших формах и пользоваться ими так же, как и любыми другими компонентами

Delphi, То есть, используя инспектор объектов, вы можете установить значения необходимых свойств, написать обработчики событий, и т. д.

Для того чтобы деинсталлировать элемент управления ActiveX, вам нужно открыть файл пакета, в который был установлен ActiveX, и убрать ненужные элементы. После этого необходимо заново откомпилировать пакет. Эти действия приведут к удалению элементов с палитры компонентов.