**Программирование в LE-технология Microsoft Windows**

**Предисловие**

Наиболее распространенным языком программирования последнего десятилетия безусловно является С. Этому способствовали такие его особенности, как лаконичность, мощность, гибкость, мобильность. Вместе с тем, стремительное усложнение приложений, для реализации которых применяются традиционные процедурно-ориентированнные языки программирования и, в частности С, заставляют говорить об определенном кризисе в их использовании, связанном прежде всего с недостаточной надежностью и выразительной способностью.

Подобных недостатков во многом лишены языки объектно-ориентированнго программирования (ООП), в сонове которыхлежит идея моделирования объектов посредством иерархически связанных классов.

Отдельно взятый класс рассматривается как совакупность множества данных и операций над ними, причем доступ к элементам данных класса возможен только посредством операций этого класса. Установление четкой взаимозависимости между данными и операциями ведет к большой целостности данных и значительно повышает надежность программ по сравнению с традиционными языками программирования. Кроме того, идея программирования с помощью классов во многом использует тот же подход, который позволяет людям формировать модели объектов реального мира.

Впервые идеи ООП были реализованы в середине 60-х годов в языке программирования Симула-67. Последний, однако, не нашел в то время широкого распространения как в силу своей относительно меньшей производительности по сравнению с традиционными языками типа FORTRAN, ALGOL, PL/1 так и, возможно, неадекватности предлагаемых средств решаемым в то время задачи. Еще одним важным ограничением для распространеия Симулы-67 стали трудности, с которыми пришлось столкнуться большинству программистов при его изучении. Дело в том, что наряду с целым рядом безусловных достоинств, идеи ООП обладают и одним существенным недостатком - они далеко не просты для понимания и особенно для освоения с целью практического использования.

С++ - это объектно-ориентированыый язык, то есть язык, позволяющий программисту оперировать объектами некоторых типов, предварительно им определенным. Название языка "С++" отражает эволюционный характер изменения языка С (запись "++", в языке С, означает, что к какой-то переменной прибавляется единица). Он имеет еще более мощные и гибкие средства для написания эффективных программ, чем С, от которого он произошел. Человек, программмирующий на традиционных языках, может просто потерять голову от тех возможностей, которые предоставляет С++.

Но не менее важным является то, что такой распространенный и универсальный язык, как С, сохранен в качестве основы. С прост, эффективен, переносим. Чего только нет в языке С: строковых данных нет, матриц нет, средств параллельного программирования тоже нет. Нет даже ввода-вывода.

Типы, операции и операторы С очень близки к тому, с чем мы имеем дело в Ассемблере,- числа, адреса, арифметические и логические действия, циклы... Кроме того, многие особенности С недвусмысленно намекаю компилятору, как сократить код и время исполнения программы. Эти характерные черты языка С позволяют написать эффективно работающий и не слишком сложный компилятор. И хотя в машинных кодах на разных компьютерах элементарные операции обозначаютс по-разному, вряд ли разработчику компилятора придет в голову интерпретировать простейшие выражения каким-нибудь оригинальным способом. Именно поэтому язык С "идет везде и на всем", программы, написанные на нем, работают эффективно, и их можно переносить с одного компьютера на другой.

**MS Windows и новый метод разработки программ.**

Одним из наиболее важных механизмов взаимодействия программ является обмен данными. В MS Windows существует несколько способов взаимодействия приложений:

- почтовый ящик;

- динамический обмен данными;

- встраивание объектов.

Специальный почтовый ящик (clipboard) Windows позволяет пользователю переносить информацию из одного приложения в другое, не заботясь об ее форматах и представлении.

В отличие от профессиональных операциональных операционных систем, где механизм обмена данными между программами доступен только программисту, в Windows это делается очень просто и наглядно для пользователя.

Механизм обмена данных между приложениями - жизненно важное свойство многозадачной среды. И в настоящее время производители программного обеспечения пришли уже к выводу, что для переноса данных из одного приложения в другое почтового ящика уже недостаточно. Появился новый, более универсальный механизм - OLE (Object Linking and Embedding)

- Встроенная объектная связь, который позволяет переносить из одного приложения в другое разнородные данные. Например, с помощью этого механизма данные, подготовленные в системе Time Line for Windows (Symantec), можно переносить в текстовый процессор Just Write ( Symantec ), а затем, скажем, в генератор приложений Object Vision (Borland). Правда, это уже нестандартное средство Microsoft Windows, но тем не менее реализация OLE стала возможной именно в Windows.

Кроме механизма почтового ящика, предназначенного, в основном, для пользователя, программисту в Windows доступны специальные средства обмена данными между приложениями.

Программным путем можно установить прямую связь между задачами, например, принимая данные из последовательного порта, автоматически помещать их, скажем, в ячейки электронной таблицы Excel, средствами которой можно тут же отображать сложные зависимости в виде графиков или осуществлять их обработку в реальном режиме времени (этот механизм носит название динамического обмена данными - Dynamic Data Exchange, DDE ).

**Основные термины**

Клиентское приложение DDE - приложение, которому необходимо установить диалог с сервером и получить данные от сервера в процессе диалога.

DDE-диалог - взаимосвязь между клиентским и серверным приложениями.

Сервер-приложение - DDE приложение, которое передает данные клиенту в процессе диалога.

DDE-Транзакция -обмен сообщениями или данными между клиентом и сервером.

Item имя - строка, идентифицирующая некоторое множество данных, которое сервер в состоянии передать клиенту в процессе диалога.

Service имя - строка, генерируемая сервером и используемая клиентом для установления диалога.

Строковый указатель - двойное слово, генерируемое операционной системой, идентифицирующее строку, передающуюся в процессе динамического обмена данными.

Topic имя - строка, которая идентифицирует тип данных, необходимых клиентскому приложению при динамическом обмене данных.

Фильтр транзакции - флаг, который препятствует передаче нежелательных типов транзакций в функцию обратного вызова.

В Microsoft Windows динамический обмен данных является формой связи, которая использует общие области памяти для обмена данными между приложениями. Приложение может использовать DDE в некоторый момент времени для передачи и получения новых данных от сервера.

Механизм DDE схож с механизмом почтового ящика, который является частью операционной системы WINDOWS. Существует лишь незначительная разница в том, что почтовый ящик, в большинстве случае, используется как буфер временного хранения информации. DDE может быть инициализирован пользователем и в большинстве случаев

продолжать работать без его вмешательства.

Библиотека DDEML обеспечивает пользователя набором средств, которые упрощают использование механизма DDE в WINDOWS приложениях. Вместо того, чтобы обрабатывать, получать и передавать DDE сообщения напрямую, приложения используют функции DDEML библиотеки. Библиотека DDEML также обеспечивает работу со строками и разделяемыми данными, генерируемыми DDE приложениями. Вместо того, чтобы использовать указатели на общие области памяти, DDE приложения создают и обмениваются строковыми указателями, которые идентифицируют строки и данные.

Уже существующие приложения, использующие протокол DDE, основанный на сообщениях полностью совместимы с теми, которые используют библиотеку DDEML. Вот почему приложение, использующее DDE-протокол могут установить диалог и выполнять транзакции с приложениями, использующими библиотеку DDEML.

**Взаимосвязь между клиентом и сервером.**

DDE возникает всегда между клиентским приложением и серверным. Клиентское приложение инициализирует обмен данными путем установления диалога с сервером и передачи транзакции. Транзакция необходима для данных и обслуживания. Сервер отвечает на транзакцию и обеспечивает клиента данными. Сервер может иметь сразу несколько клиентов в одно и тоже время, в свою очередь, клиент может получать данные сразу от нескольких серверов. Некоторое приложение одновременно может быть и клиентом и сервером. В добавок к вышесказанному, клиент и сервер могут оборвать диалог в любое удобное для них время.

DDE сервер использует три зарезервированных типа имен, расположенных иерархично: service, topic item - уникально идентифицируют некоторое множество данных, которое сервер может передать клиенту в процессе диалога.

Service имя - это строка, которую генерирует сервер в те промежутки времени, в которые клиент может установить диалог с сервером.

Topic имя - это строка, которая идентифицирует логический контекст данных. Для сервера, который манипулирует файлами, topic имена это просто названия файлов; для других серверов – это специфические имена конкретного приложения. Клиент обязательно должен указывать topic имя вместе с service именем, когда он хочет установить диалог с сервером.

Item имя - это строка, которая идентифицирует некоторое множество данных, которое сервер может передать клиенту в процессе транзакции. Например, item имя может идентифицировать ЦЕЛОЕ ( int, integer ), СТРОКУ ( string, char \* ), несколько параграфов текста, или BITMAP образ.

Все вышеуказанные имена позволяют клиенту установить диалог с сервером и получить от него данные.

**Системный режим**

Системный режим работы обеспечивает клиента всей необходимой информацией о сервере.

Для того, чтобы определить, какие серверы доступны в данный момент времени, а также какой информацией они могут обеспечить клиента, последний, находясь в начальном режиме работы, должен установить имя устройства, равное NULL. Такой шаблон диалога максимально увеличивает эффективность работы, а также работу с сервером в системном режиме. Сервер, в свою очередь, должен поддерживать нижеописанные item имена, а также другие, часто используемые клиентом:

SZDDESYS ITEM TOPICS - список item имен, с которыми может работать сервер в данный момент времени. Этот список может изменяться время от времени.

SZDDESYS ITEM SYSITEMS - список item имен, с которыми может работать сервер в системном режиме.

SZDDDESYS ITEM STATUS - запросить текущий статус сервера. Обычно, данный запрос поддерживается только в формате CF\_TEXT и содержит строку типа Готов/Занят.

SZDDE ITEM ITEMLIST - список item имен, поддерживаемых сервером в несистемном режиме работы. Этот список может меняться время от времени.

SZDDESYS ITEM FORMATS - список строк, представляющий собой список всех форматов почтового ящика, поддерживаемых сервером в данном диалоге. Например, CF\_TEXT формат представлен строкой TEXT.

Основное назначение и работа функции обратного вызова

Приложение, которое использует DDEML, должно содержать функцию обратного вызова, которая обрабатывает события, полученные приложением. DDEML уведомляет приложение о таких событиях путем посылки транзакций в функцию обратного вызова данного приложения.

В зависимости от флага фильтра транзакции, сформированного при вызове функции DdeInitialize, функция обратного вызова получает отсортированные транзакции вне зависимости от того, является ли данное приложение клиентом, сервером или тем и другим одновременно.

Параметр uType идентифицирует тип посланной транзакции в функцию обратного вызова при помощи DDEML. Значения оставшихся параметров зависят от типов транзакции. Типы транзакций будут обсуждены нами в разделе "Обработка Транзакций".

**Диалог между приложениями**

Диалог между клиентом и сервером всегда устанавливается по требованию клиента. Когда диалог установлен, оба партнера получают идентификатор, который описывает данный диалог.

Партнеры используют этот идентификатор в большинстве функций DDEML для посылки транзакций и для их обработки. Клиенту может потребоваться диалог как с одним сервером, так и с несколькими.

Рассмотрим подробно как приложение устанавливает диалог и получает информацию о уже существующих каналах связи.

**Простой Диалог**

Клиентское приложение устанавливает простой диалог с сервером путем вызова функции DdeConnect и определяет идентификаторы строк, которые содержат всю необходимую информацию о service имени текущего сервера и интересущем клиента в данный момент topic имени.

DDEML отвечает на вызов этой функции посылкой соответствующей транзакции XTYP\_CONNECT в функцию обратного вызова каждого доступного в данный момент времени сервера, зарегистрированное имя которого совпадает с именем, переданным при помощи функции DdeConnect при условии, что сервер не отключал фильтр service имени вызовом функции DdeServiceName.

Сервер может также установить фильтр на XTYP\_CONNECT транзакцию заданием соответствующего флага CBF\_FAIL\_CONNECTIONS при вызове функции DdeInitialize.

В процессе обработки транзакции типа XTYP\_CONNECT DDEML передает полученные от клиента service и topic имена серверу. Сервер должен проверить эти имена и возвратить TRUE, если он в состоянии работать с такими именами, и FALSE в противном случае. Если ни один из существующих серверов не отвечает на CONNECT-запрос клиента, функция DDeConnect возвращает ему NULL с информацией о том, что в данный момент времени НЕ возможно установить диалог.

Однако, если сервер возвратил TRUE, то диалог был успешно установлен и клиент получает идентификатор диалога - двойное слово, посредством которого и ведется обмен данными с сервером.

Затем сервер получает транзакцию вида XTYP\_CONNECT\_CONFIRM (в случае, если он НЕ описывал флаг фильтра CBF\_FAIL\_CONFIRMS при вызове соответствующей функции).

Если сервер возвращает TRUE в ответ на транзакцию XTYP\_CONNECT, DDEML посылает транзакцию вида XTYP\_CONNECT\_CONFIRM в функцию обратного вызова данного сервера. Обработав эту транзакцию, сервер может получить идендификатор диалога.

Вместо конкретного имени сервера клиент может установить шаблон диалога путем установки идентификаторов service и topic имен в NULL при вызове функции DdeConnect.

Если хотя бы один из вышеперечисленных идентификаторов равен NULL, DDEML посылает транзакцию типа XTYP\_WILDCONNECT в функцию обратного вызова всех активных в данный момент DDE-приложений (исключения составляют лишь те, кто при вызове соответствующей функции указал флаг фильтрации XTYP\_WILDCONNECT).

Любое сервер-приложение должно ответить на данную транзакцию и возвратить указатель на массив структур типа HSZPAIR, оканчивающийся нулем.

Если сервер-приложение НЕ вызывает функцию DDeNameService для регистрации собственного service имени в системе и фильтр обработки транзакций включен, то сервер НЕ получит транзакцию вида XTYP\_WILDCONNECT.

Вышеописанный массив должен содержать одну структуру для каждого service и topic имен. DDEML выбирает одну пару из массива для установления диалога и возвращает его идентификатор клиенту. Затем DDEML посылает серверу транзакцию вида XTYP\_CONNECT\_CONFIRM (исключения составляют лишь те серверы, которые при инициализации установили фильтр обработки транзакций).

Любой сервер или клиент может оборвать диалог в любое время путем вызова функции DdeDisconnect. Это означает, что партнер по обмену данными получает транзакцию типа XTYP\_DISCONNECT в функции обратного вызова (если, конечно, партнер не установил фильтр обработки транзакций вида CBF\_SKIP\_DISCONNECTIONS).

Обычно приложение реагирует на транзакцию XTYP\_DISCONNECT вызовом функции DdeQueryInfo для получения информации о прекращенном диалоге. После того, как функция обратного вызова обработала транзакцию типа XTYP\_DISCONNECT, идентификатор диалога больше не существует.

Клиентское приложение, которое получает транзакцию типа XTYP\_DISCONNECT в своей функции обратного вызова может попытаться возобновить диалог при помощи вызова функции DdeReconnect. Клиентское приложение может вызывать эту функцию только находясь внутри своей собственной функции обратного вызова.

**Сложный диалог**

Клиентское приложение может использовать функцию DdeConnectList для того, чтобы определить какие сервер-приложения существуют в системе в данный момент времени.

Клиент обязательно должен описывать service и topic имена, когда он вызывает эту функцию; это означает, что DDEML должна послать транзакцию вида XTYP\_CONNECT все функции обратного вызова всех имеющихся в данный момент сервер-приложений, чьи зарегистрированные имена совпадают с именами, указанными клиентом (исключение составляют лишь те серверы, которые фильтруют получаемые транзакции).

В добавление к вышесказанному, можно отметить, что клиент, при вызове функции DdeConnectList, может указать NULL в качестве service или topic имени, либо же сразу для обоих. Все доступные в системе серверы, чьи зарегистрированные имена совпадают с именами, указанными клиентом, отвечают на его запрос. Диалог устанавливается со всеми такими серверами, даже если в системе запущено одно и тоже сервер-приложение несколько раз.

Клиент может использовать функции DdeQueryNextServer и DdeQueryConvInfo для того, чтобы понять, какой сервер находится в списке, полученный при вызове функции DdeConnectList. DdeQueryNextServer возвращает идентификатор диалога для следующего сервера, находящегося в списке; DdeQueryConvInfo заполняет структуру CONVINFO информацией о диалоге.

Клиент может сохранить полученные идентификаторы диалогов и отказаться от просмотра оставшихся серверов в списке.

Приложение может оборвать индивидуальный диалог, находящийся в списке диалогов путем вызова функции DdeDisconnect; приложение может оборвать все диалоги, находящиеся в списке путем вызова функции DdeDisconnectList.

Обе вышеуказанные функции указывают DDEML о необходимости посылки транзакции вида XTYP\_DISCONNECT во все функции партнеров по диалогу данного приложения (в случае использования функции DdeDisconnectList будет посылаться транзакция XTYP\_DISCONNECT для каждого элемента в списке диалогов).

**Обмен данными между приложениями**

Так как DDE использует области памяти для передачи данных из одного приложения в другое, DDEML обеспечивает конечного программиста функциями, при помощи которых DDE-приложения могут создавать и обрабатывать DDE-объекты.

Весь спектр транзакций, который вызывает обмен данными, требует от приложения, экспортирующего их, создания некоторого буфера, содержащего эти данные, а затем вызова функции DdeCreateDataHandle.

Эта функция создает DDE-объект, копирует данные из буфера в этот объект и возвращает идентификатор данных для данного приложения.

Идентификатор данных-это двойное слово, которое использует DDEML для обеспечения доступа к данным в DDE-объекте.

Для того, чтобы разделять данные в DDE-объекте, приложение передает идентификатор данных DDEML, а затем DDEML передает его в функцию обратного вызова приложения, получающего данные.

Клиентское приложение получает указатель на DDE-объект путем передачи идентификатора данных функции DdeAccessData. Указатель, возвращаемый этой функцией, обеспечивает доступ к данным в формате 'ТОЛЬКО НА ЧТЕНИЕ'. Клиент должен просмотреть полученные данные при помощи этого указателя и вызвать функцию DdeUnaccessData для его уничтожения. Клиент может скопировать полученные данные в заранее приготовленный буфер посредством вызова функции DdeGetData.

Обычно, когда приложение, создающее идентификатор данных, передает его DDEML, этот идентификатор портится внутри вышеуказанного приложения. В этом нет ничего страшного, если сервер должен разделять данные только с одним клиентом. Если же сервер должен разделять данные сразу с несколькими клиентами одновременно, ему придется указывать флаг HDATA\_APPOWNED при вызове функции DdeCreateDataHandle.

Это делает возможным получение прав собственности на DDE-объект сервер-приложения и предотвращает порчу идентификатора данных DDEML. Приложение может передавать DDEML идентификатор данных любое количество раз, однако вызывать функцию DdeCreateDataHandle можно лишь однажды.

Если приложение указывает флаг HDATA\_APPOWNED в параметре atCmd при вызове функции DdeCreateDataHandle, оно обязательно должно вызывать функцию DdeFreeDataHandle для очистки памяти вне зависимости от того, передавался ли идентификатор данных DDEML или нет. Перед тем как оборвать диалог, приложение должно вызывать функцию DdeFreeDataHandle для очистки всех созданных идентификаторов, но которые так и не были переданы DDEML.

Если приложение еще не передало идентификатор DDE-объекта DDEML, то оно может добавить данные к уже существующему объекту или полностью заменить их в нем. Все эти сервисные функции обслуживаются функцией DdeAddData.

Обычно приложение использует эту функцию для новой инициализации старых не уничтоженных DDE-объектов. После того, как приложение передает идентификатор данных DDEML, DDE-объект, идентифицирующий этот идентификатор НЕ может быть изменен, однако он может быть уничтожен.

**OLE-технология**

Как видно из описанного выше протокола DDE, приложения должны обязательно знать типы передаваемых данных, уметь их обрабатывать, а в основном даже могут работать только с символьными строками. Это, конечно, не очень удобно, когда необходимо, например, создать небольшой текст с различными картинками, пиктограммами и другими наглядными или не очень иллюстрациями. В этом случае на помощь программисту приходит OLE - встраивание объектов.

Вместе с данными мы получаем машинный код, который эти данные может обрабатывать.

**Способы упорядочивания, источники и целевые документы**

При использовании OLE-технологии пользователь всегда имеет дело с одним ведущим приложением (главным) и одним ведомым (подчиненным), а точнее, с одним ведомым.

Приложение, с помощью которого получен объект для встраивания всегда играет роль подчиненного. Это особенно характерно для случаев передачи объектов при встраивании и связывании через буфер промежуточного обмена.

Часто используемые термины Приложение-источник и Целевое приложение касаются не подчинения приложений, а определяют генеалогию объектов.

Некоторые Windows-приложения могут выступать только в роли подчиненных, а некоторые только в роли ведущих. Например, Paintbrush в OLE технологии может играть только роль подчиненного приложения, служащего для создания и модификации отдельных объектов. Другие приложения, например, Write или Cardfile можно считать оправданным с точки зрения, что гораздо чаще приходится вставлять иллюстрации в сложные по структуре текст, чем текст в иллюстрации. Новые приложения, такие как Word, могут выполнять в рамках OLE обе эти функции.

Употребление термина объект считается престижным в кругах программистов, хотя часто он употребляется и не к месту. Всякий разработчик почитает своим долгом применить в своем продукте ООП без особой на то необходимости. В среде Windows в термин объект вкладывается несколько специфический смысл. Пользователя не приглашают постигать азы ООП, или заняться конструированием объектов на С++.

Когда об объектах говорят в рамках Windows, то имеют в виду возможность встраивания в некоторый документ фрагмента, порожденного другим приложением. Вот это "инородное тело" и называется объектом.

В таком подходе нет ничего нового. Когда в текст, подготавливаемый Write, вставляется рисунок из Paintbrush посредством Clipboard или таблиц Exсel, в документ, подготавливаемый в Word, то результатом действия будет как раз появления объекта.

Традиционные объекты всегда представляют собой копии. Работа с ними основывается на том, что все Windows приложения поддерживают не только свой собственный формат, но и некоторый обобщенный, стандартный, играющий роль общеизвестного международного языка. Если, например, в текстовый документ вставляется таблица из табличного процессора, то буфер промежуточного обмена преобразует ее в формат к стандартному, и тем самым обеспечивает вставку. Такая копия в текстовом редакторе по виду не отличается от оригиналу, но она недоступна для внесения изменений. Невозможно, вставив таким способом копию из Paintbrush в Write документ, изменить цвет, толщину линий или масштаб.

Новые объекты, доступные в рамках Windows 3.1 очень похожи на традиционные, но они не являются копиями - это оригиналы. Они имеются в единственном экземпляре и находятся непостредственно в целевых документах. Там они существуют одновременно в двух форматах - в стандартном и в формате приложения-источника.

Благодаря стандартному формату объект может индицироваться и сохранять в рамках целевого документа. Имеется возможность обработки объекта также, как и любого файла оригинала. Ситуация выглядит так, словно внутри объекта встроен другой. Это обеспечивает доступ к средствам обработки нового объекта (приложению-источнику) посредством простого двойного щелчка на объекте.

**Встроенные объекты**

Информация, вставленная в документ целевого приложения, представляет собой объект. Такой объект встраивается в документ, обрабатываемый ведущим приложением. Это значит, что он рассматривается как составная часть данного документа, может распечатываться и сохраняться вместе с ним. Такие объекты могут содержать информацию любого типа: текст, таблицы, графики и др.

Встроенные объекты существуют только в единственном экземпляре и только там, где они встроены - в целевом документе. Обрабатываются они своими "родительскими" программами, вызываемыми весьма эффективным способом, в отличие от традиционного.

**Связывание с родительским приложением**

Следующей весьма удобной особенностью встроенных объектов является то, что они остаются связанными с породившим их приложениями. Благодаря этому пользователь избавляется от необходимости помнить имена и директории файлов-источников. Достаточно двойного щелчка на объекте - и родительская программа запускается.

Важным достоинством подобного связывания встроенных объектов является мобильность документов. Можно легко перенести такой документ с одной машины на другую (необходимо только чтобы на них обеих была установлена оболочка и были необходимые приложения или динамические библиотеки от них). Для обработки встроенных объектов достаточно будет щелкнуть по ней дважды и на другой машине произойдет тоже самое, что и на вашей: вызовется соответствующее приложение. В этом случае необходимым условием переноса является наличие на другой машине текстового редактора Write и графического редактора Paintbrush.

При работе в рамках DDE такой перенос не возможен, точнее он будет включать в себя не только перенос самого файла-документа, но и связанных с данным файлом файлов-источников и целевых файлов - всей структуры.

**Перспективы развития OLE**

Технология OLE делает только первые шаги. Пока только некоторые Windows приложения являются OLE совместимыми. Среди утилит группы Accessories версии 3.1 такими на сегодняшний день являются только Write, Paintbrush и Cardfile. Но даже они "в своем кругу" не допускают вставки в произвольном направлении (т.е. из любой в любую другую). В настающее время речь идет о поддержке наиболее оправданного с практической точки зрения "направления встраивания" - из Paintbrush в Write и Сardfile документа.

Чтобы определить какие из приложений поддерживаю OLE интерфейс, необходимо из OLE-совместимого приложения выполнить директиву "ВСТАВИТЬ ОБЪЕКТ" в меню "Edit". В отрывшемся окне будет продемонстрирован список доступных встраиваемых объектов.

В настоящий момент многие компиляторы уже ввели поддержку OLE в свои библиотеки: Borland C++ ver4.5. Пример использования OLE технологии приведен в приложении 1. Данная программа использует созданный рисунок Paintbrush в виде файла или копирует его из Clipboard.

**Заключение**

В заключении хотелось бы отметить, что существующие способы обмена информации возникали вместе с развитием Windows. Как сама суть Windows, они являются продолжением заложенной в нее цель: cспособность работать с файлами любых форматов, на любом оборудовании. В отличие от стандартного решения, когда фирма-производитель оболочки (типа Windows) пыталась сама написать различные драйверы для поддержки устройств и различные библиотеки для поддержки форматов многочисленных файлов других пакетов, фирма Microsoft возложила эту обязанность на производителей оборудования и программного обеспечения. Таким образом, последовательное развитие Clipboard-->DDE-->OLE является продолжением воплощения идеи "сам изобрел - сам внедряй". Естесственно, наибольшие надежды сейчас возлагаются на OLE (ее новый стандарт OLE.2), так как этот стандарт позволяет включать в себя очень мощные средства, такие как Multimedia. В одном файле может находится не только текст, рисунок, а и даже целый фильм, полностью озвученный и готовый к показу.

**Список литературы**

1. Гладков С.А. Фролов Г.В. Программирование в Microsoft Windows: В 2-х частях. М.:"ДИАЛОГ-МИФИ", 1992.

2. Фойц С. Windows 3.1 для пользователя. Пер. с немецкого Киев:BHV, 1992.

3. Microsoft Windows Software Development Kit. Version 3. Programmer's Reference, Programming Tools, Windows Extensions.

4. Charles Petzold. Programming Windows. Microsoft Press.

5 Borland C++. Usres manual.