## Буфер обмена (CLIPBOARD)

В Windows предусмотрен специальный механизм обмена данными между разными приложениями, называемый буфер обмена (clipboard). Буфер обмена представляет собой буфер, в который могут быть помещены данные каким-либо приложением. Все остальные приложения Windows могут прочитать эти данные или разместить в этом буфере свои.

Для того, что бы не возникало путаницы при использовании буфера обмена, Windows предполагает применение определенных форматов данных, размещаемых в буфере.

В некоторых случаях бывает удобно просмотреть данные, размещенные в буфере обмена – для этих целей Windows содержит специальную программу просмотра содержимого буфера обмена, (Clipboard Viewer). Не надо смешивать между собой сам буфер обмена и программу его просмотра. Буфер обмена реализован несколькими функциями Windows и специальными данными.

При рассмотрении буфера обмена нам надо будет рассмотреть три вопроса:

1. как можно самим класть или читать данные из буфера обмена
2. как можно использовать буфер обмена со стандартным окном–редактором
3. как написать собственную программу просмотра содержимого буфера обмена.

Предварительно мы разберемся с некоторыми основными понятиями, связанными с применением буфера обмена.

Как мы уже сказали, Windows предполагает использование определенных форматов данных для передачи через буфер обмена. Конечно, у Вас есть возможность передавать данные в собственном формате, только для использования собственным приложением, однако рекомендуется придерживаться общепринятых стандартов, так как возможность передачи данных между самыми разнообразными приложениями является очень удобной.

Каждому применяемому формату данных буфера обмена в Windows поставлен в соответствие определенный номер. Windows определяет несколько стандартных форматов и предоставляет для них определенные символические имена:

CF\_TEXT соответствует ASCIIZ тексту

CF\_BITMAP обычный битмап

CF\_DIB битмап, независящий от устройства

CF\_PALETTE палитра (обычно применяется вместе с CF\_DIB)

CF\_METAFILEPICT метафайл

При отображении данных этих форматов в программе просмотра буфера обмена не возникает никаких проблем, так как Windows содержит все необходимые средства для отображения этих данных. Однако Вы можете класть в буфер обмена данные в собственном формате. Если Вы хотите, что бы их отображала стандартная программа просмотра, то Вы должны их объявить как

CF\_OWNERDISPLAY данные, отображаемые пользователем

В этом случае программа просмотра будет посылать специальные сообщения Вашему окну для отображения этих данных в окне.

Несколько дополнительных форматов, являясь обычными форматами данных, имеют отличные от них номера. В символических именах таких данных присутствует аббревиатура ‘DSP’

CF\_DSPTEXT соответствует ASCIIZ тексту

CF\_DSPBITMAP обычный битмап

CF\_DSPMETAFILEPICT метафайл

данные этих форматов отображаются в программе просмотра как данные соответствующих форматов, но обычно не используются другими приложениями, кроме Вашего.

Помимо рассмотренных, Windows дополнительно определяет большое количество других стандартных форматов данных, однако они используются сравнительно редко. В большинстве случаев это специфические форматы данных разных популярных программ, которые было решено включить в стандарт Windows.

При необходимости использования собственного формата данных для буфера обмена надо получить номер этого формата. Для того, что бы избежать возможных накладок, Вы должны зарегистрировать Ваш формат данных с помощью функции

UINT RegisterClipboardFormat( lpszFormatName );

для уже зарегистрированного формата Вы можете узнать его имя:

int GetClipboardFormatName( nFormat, lpsBuffer, nMaxCount );

Буфер обмена содержит не более одного блока данных каждого формата, причем все данные, находящиеся в буфере должны быть положены одним окном. Так как данные, передаваемые в буфер обмена, должны быть доступны всем приложениям, то для их передачи используются только блоки глобальной памяти.

### Запись и чтение данных из буфера обмена

Общие правила работы с буфером обмена сводятся к следующему:

1) Вся работа с буфером обмена должна проводиться за время обработки одного сообщения. Во время работы с буфером Вы не должны вызывать никаких функций, которые могут передать управление другому приложению. То есть Вы не должны использовать функций типа: DialogBox, MessageBox, GetMessage, PeekMessage.

Данные должны размещаться только в перемещаемом блоке глобальной памяти.

Когда буфер обмена получает данные, он объявляет себя владельцем этих данных, так что приложение больше не должно использовать переданные данные. Более того, эти данные нельзя удалять при завершении работы Вашего приложения - когда будет надо, буфер обмена сам удалит их.

Когда Вы читаете данные из буфера обмена, то Вы получаете хендл блока данных. Так как эти данные закреплены за буфером, то Вы не должны с ними работать, Вам необходимо скопировать их к себе.

При обмене данными с буфером обмена нельзя передавать ему запертые блоки данных, равно как нельзя оставлять их запертыми после чтения.

2) Перед началом обмена данными с буфером обмена Вы должны его открыть. Делается это с помощью функции

BOOL OpenClipboard( hWnd );

если Вы положите какие-либо данные в буфер, то окно, указанное Вами, будет считаться владельцем всех данных буфера обмена.

3) Затем Вы можете осуществить необходимые операции обмена данными. Если Вы собираетесь положить данные в буфер обмена, то Вы должны предварительно удалить все уже находящиеся в нем данные:

BOOL EmptyClipboard( void );

и только затем положить нужные данные, воспользовавшись функцией:

HGLOBAL SetClipboardData( nFormat, hGlobal );

параметр nFormat задает имя формата данных, а hGlobal является хендлом глобального блока данных. Функция возвращает Вам новый хендл этого блока данных, с помощью которого Вы можете обращаться к этим данным до закрытия буфера обмена.

Вы можете положить в буфер обмена несколько блоков данных разного формата одновременно. Так как положенные в буфер данные сохраняются там либо до его очистки, либо до завершения работы Windows, то передавать большие блоки может быть слишком накладно.

Для этого случая в Windows предусмотрен механизм передачи данных с задержкой. Вызывая функцию SetClipboardData Вы указываете вместо хендла блока данных NULL. Это означает, что данные для буфер обмена у Вас есть, но передавать Вы их будете только по требованию. Для такой передачи Вам надо будет обрабатывать три сообщения:

WM\_RENDERFORMAT nFormat 0L

сообщение требует данные для буфера обмена. При этом буфер уже открыт другим приложением, поэтому Вам открывать или закрывать его не надо. Вам надо просто вызвать функцию

SetClipboardData( nFormat, hGlobal );

передав ей хендл реального блока данных.

WM\_RENDERALLFORMATS, 0, 0L

сообщение посылается Вашему окну когда оно уничтожается, а буфер обмена содержит задержанные данные Вашего окна. Вы должны обычным образом (то есть открыть–очистить–положить–закрыть) передать все данные в буфер обмена.

WM\_DESTROYCLIPBOARD, 0, 0L

сообщение информирует Вас о том, что вызвана функция EmptyClipboard, когда буфер обмена содержит задержанные данные Вашего окна. Ваши данные больше не понадобятся, поэтому Вы можете освободить используемые структуры данных.

Если Вы собираетесь только читать данные из буфера обмена, то очищать его не надо, а получить данные нужного формата можно с помощью функции

HGLOBAL GetClipboardData( nFormat );

Функция возвращает хендл глобального блока памяти, который Вы должны скопировать к себе.

4) После завершения обмена с буфером обмена Вы должны закрыть его с помощью функции

BOOL CloseClipboard( void );

На этом заканчиваются операции обмена данными с буфером обмена.

Кроме рассмотренных, Вы можете применять еще несколько функций, облегчающих работу с буфером обмена:

BOOL IsClipboardFormatAvailable( nFormat );

Эта функция сообщает, присутствуют–ли данные нужного формата в буфере обмена. Проверку на наличие тех или иных форматов данных можно выполнить и иным способом, с помощью функции

UINT EnumClipboardFormats( nFormat );

Эта функция перебирает присутствующие форматы данных в буфере обмена и возвращает номер следующего в списке или 0. Пример применения:

UINT nFormat= 0;

while ( ( nFormat= EnumClipboardFormats( nFormat ) ) != 0 ) { // перебор форматов }

Можно узнать количество присутствующих в буфере обмена форматов данных с помощью функции

UINT CountClipboardFormats( void );

5) Сейчас мы рассмотрим правила применения форматов данных CF\_DSP... Эти данные предназначены для использования только Вашим приложениями.

Основная идея заключается в том, что приложения, читающие, скажем, формат CF\_TEXT, его и будут запрашивать у буфера обмена. При этом, даже если буфер обмена содержит данные в формате CF\_DSPTEXT, он их не передаст – номера форматов разные, поэтому считается, что Вы можете передавать данные в формате CF\_DSP... только для своего приложения, “спрятав” его от остальных.

Однако может случиться так, что одновременно два разных приложения попробуют использовать приватные данные в таком формате. Поэтому возникает необходимость научиться различать данные, положенные Вашим приложением, от данных, положенных чужим приложением. Для этого Вы можете воспользоваться функцией

HWND GetClipboardOwner( void );

которая вернет хендл окна, положившего данные. Если вы затрудняетесь по хендлу определить принадлежность окна к нужному приложению, то Вы почти наверняка знаете, к какому классу оно должно принадлежать (так как и данные, и приложение, и класс окон разработаны Вами). Поэтому Вы можете узнать еще и имя класса окна:

int GetClassName( hWnd, lpsBuffer, nMaxCount );

### Буфер обмена и стандартное окно–редактор

Часто приходится использовать буфер обмена совместно с обычным окном редактора. Больших сложностей здесь нет, так как такое окно автоматически поддерживает операции обмена данными с буфером в формате CF\_TEXT. Единственное, что нам надо сделать – научиться передавать редактору команды для осуществления этого обмена.

Это обычно приходится делать для того, что бы добавить меню, содержащее пункты Cut–Copy–Paste. Так как сам редактор, будучи дочерним окном, не имеет меню, то меню добавляется к нашему родительскому окну. То есть команды, полученные от меню, надо как-то передать в окно редактора.

При этом заодно приходится разрешать/запрещать отдельные пункты меню в зависимости от наличия выделенного текста в редакторе (Cut, Copy) и наличия нужных данных в буфере обмена (Paste).

Для передачи обычному редактору команд для обмена данными с буфером служат сообщения

WM\_PASTE 0 0L

вставить текст из Clipboard в текущую позицию каретки

WM\_COPY 0 0L

скопировать выделенный текст из редактора в Clipboard

WM\_CUT 0 0L

скопировать выделенный текст и удалить его из редактора

Мы должны просто послать нужное сообщение редактору и он выполнит все остальное. Обратите внимание на то, что эти сообщения не являются специфичными для редактора – они имеют префикс WM\_ – то есть это сообщения для обычного окна.

Если Ваше окно будет поддерживать работу с буфером обмена, то настоятельно рекомендуется поддерживать эти сообщения.

Для того, что бы разрешать или запрещать нужные пункты меню можно воспользоваться функцией IsClipboardFormatAvailable для разрешения/запрещения операции Paste и передачей сообщения EM\_GETSEL для выяснения возможности операций Cut и Copy.

### Просмотр данных, находящихся в буфере обмена

Если Вы вводите новые форматы данных в буфер обмена, то вам может понадобиться новая программа просмотра данных буфера обмена, которая позволит просматривать Ваши данные. В Windows принято, что программа просмотра должна обновлять отображаемую информацию при смене данных в буфере обмена. Для этого программа просмотра получает сообщение

WM\_DRAWCLIPBOARD 0 0L

Однако одновременно может работать несколько программ (окон) просмотра. При этом возникает необходимость где-то удерживать список всех таких программ (окон) и посылать им всем соответствующие сообщения. Для этого организуется цепочка программ просмотра, которую они сами поддерживают в корректном состоянии.

В какой-то степени это похоже на обработку прерываний DOS - система удерживает хендл только первого окна просмотра, тот - хендл следующего и так далее.

Когда Вы запускаете свою программу просмотра, она регистрируется в качестве программы просмотра буфера обмена с помощью функции

HWND SetClipboardViewer( hWnd );

при этом он становится на первое место в цепочке, а функция SetClipboardViewer возвращает хендл следующего за ним (или 0, если других программ (окон) просмотра нет). Этот хендл должен быть сохранен.

Далее, при обновлении данных в буфере обмена, Ваше окно получает сообщение WM\_DRAWCLIPBOARD. Обычная обработка этого сообщения:

case WM\_DRAWCLIPBOARD: if ( hWndNextViewer ) PostMessage( hWndNextViewer, wMsg, wParam, lParam ); InvalidateRect( hWnd, NULL, TRUE ); return 0;

В результате обработки этого сообщения окно программы просмотра перерисовывается, и такое же сообщение получает другая программа.

Во время нормальной работы какая-либо из программ просмотра заканчивает функционирование прежде остальных. При этом цепочка программ просмотра изменяется. Для того, чтобы корректно изменить цепочку, применяется функция

BOOL ChangeClipboardChain( hWndViewer, hWndNextViewer );

Что значит: вместо окна hWndViewer будет использоваться окно hWndNextViewer. Эта функция обычно вызывается при обработке сообщения WM\_DESTROY. В результате выполнения этой функции первая программа просмотра в цепочке получит сообщение (WM\_CHANGECBCHAIN, hWndRemoved, hWndNext). Это сообщение обрабатывается так:

case WM\_CHANGECBCHAIN: if ( wParam == hWndNextViewer ) { hWndNextViewer= LOWORD( lParam ); } else if ( hWndNextViewer ) { PostMessage( hWndNextViewer, wMsg, wParam, lParam ); } return 0;

В результате такой обработки, если наше окно стоит в цепочке до удаляемого, но не прямо перед ним, оно передаст сообщение дальше; если оно стоит прямо перед удаляемым, то оно исправит свой хендл следующего окна просмотра и не пустит сообщение дальше, так как цепочка уже исправлена.

Отображение данных программой просмотра происходит также, как и их обычное чтение (открыл–прочитал–закрыл).

## Динамический обмен данными (DDE)

Пока что мы рассмотрели только один способ обмениваться данными между разными приложениями - буфер обмена. Однако этот способ не всегда эффективен. В большинстве случаев он ограничен применением в редакторах.

Однако необходимость обмена данными между разными приложениями реально существует, поэтому Windows содержит еще одно средство для такого обмена — DDE (Dynamic Data Exchange).

DDE основан на использовании сообщений для передачи данных между двумя приложениями. При этом одно из приложений (обычно содержащее данные) называется сервером (server), а другое (обычно требующее данных) – клиентом (client). В более общем виде: клиент выступает в роли активного приложения, требующего, что бы его запросы были обслужены сервером. Сам процесс обмена данными средствами DDE между клиентом и сервером называется DDE–разговором (DDE–conversation).

Обычно протокол обмена данными между клиентом и сервером выглядит примерно следующим образом:

* клиент передает всем доступным приложениям сообщение о том, что ему надо.
* если в системе находится подходящий сервер, он отвечает клиенту о возможности обслуживания.
* клиент посылает запрос(ы) серверу для выполнения требований.
* в некоторых случаях сервер может информировать клиента об изменении или наличии тех или иных данных. Для этого клиент должен подписаться (advise) на необходимые данные.
* обмен между клиентом и сервером может продолжаться до тех пор, пока один из них не потребует прекращения.

В процессе DDE–разговора происходит обмен сообщениями между двумя приложениями. Понятно, что адресатами таких сообщений являются не сами приложения, а их окна. В большинстве случаев для нормального ведения DDE–разговора создаются специальные скрытые (невидимые) окна, которые и осуществляют обмен сообщениями между собой. Конечно, в DDE могут участвовать окна одного приложения, не только разных, но надо учесть, что DDE для обмена данными в пределах одного приложения является неэффективным. Так как процессы подготовки и приема данных могут занимать значительное время, то требуется, чтобы для обмена сообщениями использовалась функция PostMessage, а не SendMessage (кроме установления DDE–разговора). Это, правда, создает дополнительную сложность — необходимость синхронизации действий разных окон между собой.

В Windows содержится все необходимое для организации DDE, причем в двух экземплярах. Помимо старого способа, существующего с первых версий Windows, в Windows версии 3.1 была добавлена специальная библиотека DDEML (DDE Management Library), являющаяся “надстройкой” над старым способом и предназначенная для формализации протоколов обмена. Существенная разница между этими способами связана с тем, что для организации DDE старым способом вы должны предусмотреть обработку специальных сообщений необходимыми Вам окнами, а библиотека DDEML сама создает необходимые скрытые окна и организует обмен сообщениями; при этом для взаимодействия с Вашим приложением DDEML будет вызывать специально разработанную Вами CALLBACK процедуру (не являющую оконной процедурой).

Считается, что при использовании DDEML несколько упрощается написание приложений, так как она берет на себя вопросы синхронизации обмена. Однако на практике не было замечено реального упрощения работы в связи с применением библиотеки. Более того, исходный текст может оказаться даже больше, чем при использовании старого метода.

Поэтому мы будем рассматривать только старый способ организации DDE. Предварительно мы введем несколько терминов, используемых в DDE. Когда клиент начинает DDE или требует данные, он посылает серверу спецификацию того, что он требует.

Эта спецификация состоит из 3 пунктов:

* имя приложения — application (в DDEML называется сервис (service)); так как в большинстве случаев в документации применяется термин service, то его мы и будем использовать дальше. Хотя, надо отметить, этот параметр обычно задает условное имя приложения–сервера.
* тему DDE–разговора — topic
* требуемые данные — item

Имя сервиса и тема DDE–разговора используютсяпри установлении связи. В результате этого в системе должно быть установлен обмен данными между одной или несколькими парами окон, которые поддерживают данные сервис и тему. Одно из окон в каждой паре будет являться клиентом, а другое — сервером. В процессе дальнейшего DDE–разговора может происходить обмен данными, имя которых задается отдельно и является возможным именем данных для данного сервиса и темы.

Эти три имени представлены в виде атомов (ATOM), поэтому нам надо разобраться с атомами и правилами их применения перед продолжением разговора о DDE.

### Атомы

Атомами в Windows называются нумерованные строки текста, хранимые в специальной таблице. Максимальный размер строки 255 символов. При помещении строки в таблицу ей присваивается уникальный номер, который собственно и используется вместо самой строки. Часто атомом называют именно эти номера, а строки — именами атомов. Обе платформы (Windows 3.x и Win32) используют для задания атомов 16-ти разрядные числа, называемые ATOM, что позволяет в одном двойном слове передавать до двух атомов.

Если добавлять две одинаковых строки в таблицу атомов, то они получат одинаковый номер. На этом основан один из способов сравнения строк — добавить обе строки в таблицу и сравнить атомы (не забудьте после этого удалить добавленный атом, даже если это было надо только для проверки).

Для каждого приложения доступны две таблицы атомов - локальная и глобальная. Так как DDE поддерживается между разными приложениями то, очевидно, должна применяться только глобальная таблица атомов.

Для работы с глобальными атомами предназначены следующие функции:

ATOM GlobalAddAtom( lpszName ); UINT GlobalGetAtomName( aAtom, lpsBuffer, nMaxCount ); ATOM GlobalFindAtom( lpszName ); ATOM GlobalDeleteAtom( aAtom );

С помощью этих функций можно добавить атомы, узнать имя конкретного атома, найти нужный атом в таблице и удалить атом. Причем, сколько раз добавлялся атом с одним именем, столько раз он должен быть удален, что бы он действительно был удален из таблицы.

Для работы с локальными атомами используются аналогичные функции, но не имеющие в начале слова “Global”. Помимо этого для работы с локальной таблицей атомов добавлена еще одна функция:

ATOM AddAtom( lpszName ); UINT GetAtomName( aAtom, lpsBuffer, nMaxCount ); ATOM FindAtom( lpszName ); ATOM DeleteAtom( aAtom );

BOOL InitAtomTable( UINT cTableEntries );

Которая позволяет указать максимальное число строк, помещаемых в локальную таблицу атомов. По умолчанию это число равно 37 (размер глобальной таблицы атомов изменить нельзя). При работе с локальной таблицей атомов удалять атомы в конце работы приложения необязательно — при завершении работы всего приложения таблица будет автоматически удалена.

При обмене данными посредством DDE происходит последовательная посылка сообщений от одного приложения к другому и наоборот. Одна DDE транзакция обычно состоит из двух — трех последовательных сообщений. В этих сообщениях часто передаются необходимые имена, представленные в виде глобальных атомов. При этом возникает необходимость так организовать обмен сообщениями, что бы обеспечить удаление всех вновь добавляемых атомов. Для этого принято такое решение: приложение, посылающее атомы вместе с сообщение должно эти атомы создать, а затем проверить код завершения функции PostMessage — если посылка прошла успешно, то атом должен быть освобожден принимающим приложением, а если при посылке возникла ошибка, то атом должно удалить посылющее приложение. В некоторых случаях атом, полученный приложением с сообщением, может быть послан вместе с ответным сообщением назад (кроме сообщения WM\_DDE\_INITIATE, о чем см. ниже). В этом случае не обязательно атом удалять а потом создавать снова, можно использовать полученный; однако необходимо убедиться, что он был послан с ответным сообщением и, если посылка не состоялась, его надо удалить.

### Особенности задания параметра lParam сообщений DDE

Следует учесть некоторые различия в применении параметра lParam сообщений DDE на 16–ти и 32–х разрядных платформах (Windows 3.x и Win32). Дело в том, что первоначально этот параметр использовался для передачи двух значений, часто атома имени данных и хендла глобального блока. В случае 32–х разрядной платформы хендл блока сам занимает двойное слово и разместиться в lParam совместно с атомом имени данных не может.

Для решения этой проблемы были введены специальные функции, осуществляющие “упаковку” двух указанных значений (называемых младшим и старшим) в одно двойное слово. Фактически можно считать, что выделяется специальная структура, содержащая пару полей, идентификатор которой используется в качестве lParam.

Так как разные сообщения DDE используют lParam различным образом, то и упаковка данных осуществляется не для каждого сообщения, так что для каждого сообщения надо проверять необходимость использования этого механизма, а для сообщения WM\_DDE\_ACK надо еще проверить, в ответ на какое сообщение он послан. Для работы с “упакованными” данными необходимо применять специальные функции:

LONG PackDDElParam( UINT uMsg, UINT uLow, UINT uHigh ); LONG UnpackDDElParam( UINT uMsg, LONG lParam, PUINT puLow, PUINT puHigh ); LONG FreeDDElParam( UINT uMsg, LONG lParam ); LONG ReuseDDElParam( LONG lParam, UINT uMsgIn, UINT uMsgOut, UINT uLow, UINT uHigh );

Функция PackDDElParam упаковывает указанные данные в промежуточную структуру, хендл которой возвращается в виде LONG; возвращаемое значение может быть использовано только для передачи данных с помощью функции PostMessage и только для некоторых сообщений DDE. Функция UnpackDDElParam выполняет обратную операцию, но промежуточная структура данных при этом сохраняется. Для того, что бы ее удалить необходимо использовать FreeDDElParam. Последняя функция ReuseDDElParam может применяться для использования одной структуры при повторной передаче данных или ответе на принятое сообщение. В качестве параметра lParam функции PostMessage необходимо указывать возвращаемое, а не первоначальное значение.

В описании параметров сообщений мы сохраняем прежний синтаксис, причем lParam может по–прежнему задаваться в виде двух компонент старший & младший. Единственное отличие, что под старшим и младшим компонентом не надо подразумевать слова, а для получения этих компонентом может потребоваться функция UnpackDDElParam, о чем в описании сообщения будет сделана специальная оговорка. В редких случаях могут даваться два описания параметров, для Windows 3.x и для Win32. Для различия этих вариантов в описании параметров сообщения могут быть добавлены специальные пояснения

(Packed Win32) — если параметр упаковывается в случае платформы Win32

(Windows 3.x) — если описание применяется только в случае Windows 3.x

(Win32) — если описание применяется только в случае Win32

### DDE, начало обмена и его завершение

Когда мы говорили об общем виде протокола DDE, мы отметили, что он основан на обмене сообщениями. В большинстве случаев сообщения посылаются (post) а не передаются (send). При этом приложение, посылающее сообщение не имеет возможности дождаться обработки этого сообщения. Поэтому те данные, которые посылаются одним приложением, должны быть уничтожены другим, что бы они не оставались в памяти. Единственное исключение из этого правила — сообщения WM\_DDE\_INITIATE и ответ на него (WM\_DDE\_ACK) которые всегда передаются, а не посылаются.

При начале DDE–разговора посылается сообщение

WM\_DDE\_INITIATE hWnd aTopic & aService

это сообщение используется для начала DDE–разговора. Параметр wParam содержит хендл пославшего окна, а lParam в младшем слове — атом aService, а в старшем — атом aTopic, задающие, соответственно, сервис и тему DDE–разговора. Нулевые значения атомов указывают соответственно на любую поддерживаемую тему и любой сервис. Передача: Когда клиент вызывает функцию SendMessage для установления DDE–разговора, он создает необходимые атомы, а после возврата из функции SendMessage он обязан эти атомы удалить. Получение: сервер, получивший это сообщение, и поддерживающий указанные сервис и тему отвечает сообщением WM\_DDE\_ACK. При этом он обязан создать заново требуемые атомы — использовать атомы, полученные с сообщением запрещено. Значения атомов aTopic и aService равные 0 указывают, соответственно, на любую поддерживаемую сервером тему и сервис. В этом случае сервер должен ответить столько раз, сколько подходящих тем и сервисов он поддерживает.

Это сообщение передается всем приложениям (в качестве хендла окна–получателя сообщения используется HWND\_BROADCAST или -1). Тот сервер, который поддерживает данную тему и сервис должен ответить на этот запрос передачей сообщения

WM\_DDE\_ACK hWnd aTopic & aService

сообщение, подтверждающее принятое прежде сообщение. Параметр wParam содержит хендл пославшего окна, а lParam используется разными способами, в зависимости от того, какое сообщение вызвало это подтверждение. В зависимости от значений параметров сообщения рассматривают положительные и отрицательные подтверждения. При ответе на WM\_DDE\_INITIATE младшее слово lParam содержит атом сервиса, а старшее — атом темы, при этом WM\_DDE\_ACK не посылается, а передается с помощью функции SendMessage, причем оно рассматривается только как положительное, так как в качестве отрицательного ответа используется отсутствие этого сообщения. Если WM\_DDE\_ACK получено в ответ на WM\_DDE\_INITIATE, то для обоих платформ — Windows 3.x и Win32 оно используется одинаково; а если оно получено в ответ на какое–либо иное сообщение, то параметр lParam будет использован для передачи “упакованного” значения. Получение: приложение–клиент, получившее сообщение WM\_DDE\_ACK обязано удалить все сопровождающие его атомы. Передача: при ответе на WM\_DDE\_INITIATE запрещено использовать полученные атомы, сервер обязан создать необходимые для ответа атомы и послать их (так как на одно WM\_DDE\_INITIATE может ответить несколько серверов, а при обработке WM\_DDE\_ACK клиент обязан удалять все атомы).

Если WM\_DDE\_ACK передается в ответ на сообщение WM\_DDE\_INITIATE, то в lParam содержатся такие–же данные, что и у сообщения WM\_DDE\_INITIATE. Это считается положительным подтверждением. Если ответ отрицательный, то подтверждение при инициализации просто не выдается.

В большинстве случаев сервер, отвечающий на WM\_DDE\_INITIATE, создает специальное окно для каждого DDE–разговора. Это связано с тем, что при дальнейшем обмене данными сервис и тема указываться не будут, а сам DDE–разговор определяется фактически окном–клиентом и окном–сервером. Одна такая пара окон обменивается данными только в рамках указанных при установлении DDE–разговора темы и сервиса. Только самые простые DDE–серверы могут обходиться одним окном, но при этом они в данный момент времени могут работать только с одним клиентом.

Внимание! При установлении связи с каким–либо сервером Вы можете получить несколько сообщений WM\_DDE\_ACK от разных серверов, поддерживающих указанные сервис и тему. Если в сообщении WM\_DDE\_INITIATE вы указали любую поддерживаемую тему и/или любой возможный сервис, то практически всегда отвечают сразу несколько серверов. В этом случае Вы должны создавать список всех тем и сервисов, с которыми устанавливается связь данной операцией (не забудьте в конце закрыть все начатые DDE–разговоры), а если вы собираетесь работать только с одним из ответивших серверов, то со всеми остальными вы должны завершить DDE–разговор посылкой WM\_DDE\_TERMINATE.

Замечание. Согласно документации сервер, получивший WM\_DDE\_INITIATE с указанием любой поддерживаемой темы или сервиса, обязан отправить в ответ столько сообщений WM\_DDE\_ACK, сколько тем и сервисов он поддерживает. На самом деле многие DDE серверы этого не делают, как, например, Microsoft Internet Explorer. Такие серверы отвечают только на точно указанные сервис и тему. Это может быть оправдано, если сервер поддерживает значительное число тем или сервисов. При ответе на WM\_DDE\_INITIATE уже устанавливаетя DDE–разговор, для которого сервер создает отдельное окно и, соответственно, расходует ресурсы. Если сервер отвечает на целый список тем и/или сервисов, то создается много окон, что, скорее всего, излишне. Строго говоря, указывать любую тему или сервис надо только в случае реальной необходимости, так как одним таким запросом могут быть установлены несколько десятков DDE–разговоров сразу (так, например, один только Netscape Navigator отвечает примерно 3 десятками поддерживаемых им тем).



Начало DDE–разговора - это единственный случай, когда сообщения DDE передаются с помощью SendMessage, а не PostMessage. Это необходимо для нормального начала обмена.

Сейчас мы рассмотрим небольшой пример взаимодействия двух приложений, клиента и сервера при начале DDE–разговора.

В данном примере рассмотрен самый простой случай, когда связь устанавливается только с одним сервером и для одной темы. При этом можно обойтись без создания списка сервисов и тем, с которыми устанавливаются соединения.

Считается, что при инициализации DDE–разговора, клиент должен получить ответ (или убедится в его отсутствии) немедленно. Этого можно достичь только используя передачу, а не посылку сообщений. В этом случае подтверждение приходит в то время, пока клиент ожидает завершения работы процедуры SendMessage. Для нормального продолжения DDE клиент должен запомнить хендл сервера (а сервер - хендл клиента).

Так как возможен случай, что на запрос клиента ответят два и более серверов, то надо предусмотреть либо отказ от установления более чем одного соединения (как в приведенном примере), либо организовать DDE сразу с несколькими серверами.



Если клиент получил положительный ответ от сервера, то он может начать обмен данными. Этот обмен будет продолжаться до тех пор, пока оба приложения не обменяются сообщениями

WM\_DDE\_TERMINATE hWnd 0L

сообщение оканчивает DDE–разговор. Параметр hWnd является хендлом пославшего окна. При этом уничтожаются вспомогательные структуры и обнуляются переменные.



Послать WM\_DDE\_TERMINATE может как клиент, так и сервер. Оба они должны быть готовы к приему такого сообщения от напарника.



### Обмен данными между клиентом и сервером

Обмен данными между клиентом и сервером может происходить по нескольким различным сценариям. Всего можно выделить три способа получения данных от сервера и еще один способ, предназначенный для передачи данных от клиента к серверу.

В DDE различают три способа получения данных от сервера, называемых видами связи. Эти три вида связи называются холодная, теплая и горячая. Коротко поясним различия этих видов связи:

* холодная связь — cold link

обмен данными происходит только по запросу клиента. Сервер посылает в ответ данные или отрицательное подтверждение.

* горячая связь — hot link

клиент “подписывается” на периодическое получение данных от сервера, после чего сервер начинает передавать данные клиенту, как только в этом возникает необходимость. Для завершения горячей связи клиент должен сообщить об этом серверу.

* теплая связь — warm link

клиент, как и при горячей связи, подписывается на получение обновленных данных. Однако сервер передает не данные, а только сообщения о том, что у него есть данные для клиента. Клиент может затребовать данные у сервера в любой удобный для него момент.

Последние два вида связи (теплая и горячая) называются иногда постоянной (permanent) связью.

Передача данных от клиента к серверу осуществляется только одним способом, по инициативе клиента, который передает серверу соответствующее сообщение, содержащее посылаемые данные.

Когда два приложения обмениваются данными друг с другом, они передают друг другу хендлы блоков данных и атомы. Для того, что бы эти данные были доступны обоим приложениям, они должны быть глобальными. Однако надо учесть, что обычно глобальные блоки данных связаны с тем приложением, которое их создало, то есть при завершении приложения эти данные автоматически уничтожаются. Так как процесс DDE–разговора является асинхронным, то необходимо обеспечивать сохранность данных независимо от существования создавшего их приложения. Для этого глобальные блоки данных должны быть разделяемыми — при их выделении надо указывать флаг GMEM\_DDESHARE (или GMEM\_SHARE, который является синонимом).

В случае платформы Win32 используются прежние функции для выделения глобальных блоков данных, несмотря на то, что блоки выделяются только в локальном для каждого процесса виртуальном адресном пространстве. Система автоматически осуществляет передачу данных из адресного пространства одного процесса в адресное пространство другого процесса при передаще соответствующих сообщений.

Особое внимание надо уделить вопросу освобождения ресурсов, так как атомы и передаваемые блоки данных сами не уничтожаются, даже если все участвующие в DDE приложения завершили работу. Все созданные объекты должны быть обязательно уничтожены, независимо от исхода операции. Сложности связаны с тем, что один и тот–же объект может быть уничтожен либо клиентом, либо сервером, в зависимости от протекания процесса обмена, а, кроме того, в процессе обмена могут присходить различные ошибки (например, функция PostMessage не может послать сообщение).

#### “Холодная” связь — cold link

При холодной связи обмен данными активируется клиентом. Для этого клиент посылает специальное сообщение, в ответ на которое сервер отправляет данные (если может), либо сообщение об ошибке:

WM\_DDE\_REQUEST hWnd aItem & cfFormat

это сообщение является требованием передачи данных. Параметр wParam является хендлом пославшего сообщение окна, а lParam содержит в младшем слове номер формата данных (номера те–же, что используются буфером обмена), а в старшем - атом имени данных. Платформы Win32 и Windows 3.x используют это сообщение одинаково.

В ответ на это сообщение сервер может ответить либо сообщением WM\_DDE\_DATA, если он имеет необходимые данные и может их послать, либо “отрицательным” сообщением WM\_DDE\_ACK, указывающим, что сервер не может послать требуемые данные.

WM\_DDE\_DATA hWnd aItem & hData (Packed Win32)

Это сообщение передает данные клиенту. Младший компонент lParam содержит хендл глобального разделяемого блока данных hData. Этот параметр может быть равен 0, что реально встречается только в случае “теплой” связи (см. ниже). Передаваемый блок данных должен начинаться со структуры DDEDATA, флаг fAckReq которой указывает на необходимость отправки подтверждения о получении данных. Передача: сервер не может сбрасывать одновременно оба бита fRelease и fAckReq в 0. То есть корректными являются только три возможных комбинации fRelease и fAckReq, при которых можно определить, кто должен освобождать блок данных — клиент или сервер. Освобождение ресурсов: атом aItem должен удаляться, если только он не посылается с ответным сообщением WM\_DDE\_ACK (если бит fAckReq заголовка данных равен 1, то есть требуется ответ). Освобождение блока данных зависит от установки битов fRelease и fAckReq заголовка данных:

|  |  |  |
| --- | --- | --- |
| fRelease | fAckReq |  |
| 0 | 1 | блок данных всегда освобождается сервером при получении ответа от клиента (считается, что к моменту получения сервером подтверждения клиент уже прочитал требуемые данные и они ему больше не нужны). |
| 1 | 0 | блок данных всегда освобождается клиентом, сервер ответа не получает (так клиент может некоторое время сохранять полученные данные независимо от течения DDE–разговора). |
| 1 | 1 | при успешном чтении блок данных освобождается клиентом, а в случае ошибки (то есть при получении сервером отрицательного подтверждения) блок освобождается сервером. Этот вариант близок к предыдущему, за исключением того, что за ошибочно полученный блок клиент ответственности не несет. |

Структура DDEDATA содержит следующие данные:

typedef struct tagDDEDATA { WORD unused:12, fResponse:1, // 1 в ответ на WM\_DDE\_REQUEST, 0 — WM\_DDE\_ADVISE fRelease:1, // данные должны быть удалены после получения reserved:1, fAckReq:1; // 1 — необходимо послать подтверждение о приеме short cfFormat; // формат данных BYTE Value[1]; // сами данные } DDEDATA;

Внимание! в документации по Windows 3.x SDK и Win32 SDK содержалась ошибка — там были перепутаны пояснения к полям fResponse и fAckReq, так что описание структуры DDEDATA противоречило описанию сообщения WM\_DDE\_DATA. В документации, сопровождающей компилятор Microsoft Visual C++ v4.0 эта ошибка была исправлена.

Заметьте, что сервер может потребовать подтверждения о приеме посланных данных. Для этого он устанавливает бит fAckReq в заголовке переданных данных равным 1. В таком случае необходимо послать ответное сообщение WM\_DDE\_ACK (положительное или отрицательное, если возникла какая-либо ошибка).



Если сервер не имеет нужных данных, то вместо WM\_DDE\_DATA клиент получит WM\_DDE\_ACK (отрицательное)



WM\_DDE\_ACK hWnd aItem & wStatus (Packed Win32)

Младший компонент lParam содержит информацию о посылаемом подтверждении. Младший байт этого слова содержит код, возвращаемый приложением, старший байт содержит два значащих бита 0x80 и 0x40. Часто этот параметр представляется в виде структуры DDEACK. Старшее слово lParam представляет собой атом, идентифицирующий данные. Согласно документации этот атом обязательно должен быть удален при обработке этого сообщения. В случае платформы Win32 параметр lParam используется для передачи “упакованного” значения; при этом надо помнить, что если WM\_DDE\_ACK приходит в ответ на WM\_DDE\_INITIATE, то параметр lParam содержит обычные, не упакованные данные.

Структура DDEACK содержит следующие данные:

typedef struct tagDDEACK { WORD bAppReturnCode:8, // код, определяемый приложением reserved:6, fBusy:1, // 1 — сервер занят и не может обработать запрос fAck:1; // 1 — сообщение было получено приложением } DDEACK;

В зависимости от значений битов fBusy и fAck различают следующие возможные виды ответов:

* fAck = 1 fBusy = 0 — положительное подтверждение
* fAck = 0 fBusy = 0 — отрицательное подтверждение
* fAck = 0 fBusy = 1 — сервер занят, либо не может обработать запрос в разумные сроки (что есть “разумные” сроки — не поясняется)
* Согласно документации установка обеих флагов fAck и fBusy одновременно должна быть исключена.

Вместо структуры DDEACK часто используют просто битовые шаблоны — 0x8000 соответствует положительному подтверждению, 0 — отрицательному и 0x4000 — сервер занят (вообще говоря, это тоже отрицательное подтверждение, хотя запрос можно попробовать повторить).

Дополнительно ситуация осложняется тем, что в процессе DDE–разговора применяется посылка, а не передача сообщений. То есть между посылкой сообщения серверу и получения от него ответа может пройти значительное время, в течении которого необходимо обрабатывать остальные поступающие сообщения, система при этом может переключаться на другие приложения и, что самое неприятное, в это время либо клиент, либо сервер могут вообще завершить работу. В этой ситуации приходится вставлять специальные фрагменты кода, осуществляющие ожидание того или иного сообщения в течении указанного промежутка времени и принимать специальные меры, если ответа слишком долго нет.

// // wait for time out // static BOOL TimeOut( HWND hWnd, UINT wMsg, UINT uTimeOut ) { DWORD dwTime; MSG msg;

// FALSE design normal termination // TRUE design time-out or negative WM\_DDE\_ACK dwTime= GetCurrentTime(); while ( dwTime + uTimeOut > GetCurrentTime() ) { if ( PeekMessage( &msg, hWnd, NULL, NULL, PM\_REMOVE ) ) { // обработать полученное сообщение TranslateMessage( &msg ); DispatchMessage( &msg ); if ( msg.message == WM\_DDE\_ACK ) { // check for .fAck return LOWORD( msg.lParam ) & 0x8000 ? FALSE : TRUE; } else if ( msg.message == wMsg ) return FALSE; } }

return TRUE; }

Обычно процедура, ожидающая нужное сообщение, выглядит несколько иначе. Это связано с желательным отображением курсора ожидания (в виде песочных часов), а также с обработкой сообщений, извлеченных из очереди. Во многих случаях нормальная обработка сообщений не сводится только к функциям TranslateMessage и DispatchMessage, но также включает поддержку акселераторов, немодальных диалогов и пр. Помимо этого следует учесть, что в случае платформы Windows 3.x размер очереди сообщений приложения сравнительно невелик — так что при извлечении сообщений, направленных только конкретному окну, очередь может легко переполниться.

Еще одно замечание касается времени ожидания ответа от сервера. Этот промежуток не надо назначать очень большим, но в то же время он должен быть достаточным, что бы даже на медленных машинах уверенно получать ответ от сервера. При избыточно большом времени ожидания ответа может возникать впечатление “зависания” приложения, пока оно ждет ответа. Обычно человек ожидает от компьютера почти мгновенной реакции или, в худшем случае, явно видимой работы. Если программа останавливается на срок более 10–15 секунд, то уже возникает беспокойство о ее работе.

Практически приемлемым является интервал порядка 5–30 секунд, так как на средней машине ответ может задерживаться до 5–10 секунд при нормальной работе и, соответственно, на медленных машинах интервал следует увеличить примерно до тридцати секунд. (Различие между быстрыми и медленными машинами весьма относительно — с ростом мощности компьютеров растет нагрузка на них, поэтому время выполнения более–менее значительных операций снижается сравнительно медленно). Возможным решением является двухэтапное ожидание — первый этап до 3–6 секунд проходит как обычно, а по истечении этого интервала отображается окно диалога с кнопкой “Cancel”, после чего ожидание продолжается либо до получения ответа, либо до отмены ожидания пользователем. Это позволяет пользователю оценить занятость системы и, при необходимости, продолжить ожидание в разумных пределах.

Задача определения времени ожидания ответа от сервера является достаточно важной из–за необходимости освобождать занятые ресурсы. Фактически, при обмене сообщениями возникает следующая ситуация: при посылке каких либо разделяемых объектов от одного приложения к другому (это могут быть атомы или глобальные блоки данных) необходимо эти объекты удалять после их использования. Однако тут надо учесть следующие нюансы:

1. если функция PostMessage не может послать сообщение, объекты можно удалять сразу, так как второе приложение их не получит.
2. если сообщение послано, то момент уничтожения объектов выбирается в соответствии с описанием сообщения в документации. Так, например, атомы обычно удаляются по следующей схеме
   1. при получении посланного запроса проверяется необходимость ответа, если ответ нужен, то атом передается вместе с ответом и уничтожается приложением, пославшим запрос
   2. если ответ нужен, но его послать не удается, то атом уничтожается получившим запрос приложением
   3. если ответ не нужен, то атом опять–же уничтожается получившим запрос приложением
3. посланное сообщение может “не дойти” до адресата (например, если окно–получатель будет закрыто уже после того, как к нему в очередь попадет это сообщение, но до его обработки). При этом ответа не будет, а приложение–получатель освободить ресурсы не сможет.
4. посланное сообщение может быть успешно принято, а при его обработке, но уже после освобождения полученных ресурсов, получатель не сможет отправить обратно необходимого подтверждения.

В двух последних случаях вопрос освобождения ресурсов остается открытым, так как приложение, пославшее запрос, не может получить информацию о результатах его выполнения, и, соответственно, о необходимости освобождения данных. При этом можно слегка изменить рассмотренное ранее “двухэтапное” ожидание, добавив в диалог кнопку или флажок освобождения ресурсов. Обычно пользователь может оценить причину задержки ответа — сбой в работе, либо досрочное завершение работы приложения и момент возникновения этой ситуации — до обработки или после. По крайней мере, если ситуация будет повторяющейся, то пользователь сможет подобрать наиболее “дешевый” выход из нее.

#### “Горячая” связь — hot link

Протокол при горячей связи. После инициации обмена клиент посылает запрос WM\_DDE\_ADVISE, указывая желаемые данные. Сервер отвечает о возможности горячей связи, и, обычно, сразу же посылает первые данные, для инициации клиента. Таким образом осуществялется как–бы “подписка” на получение необходимых данных при их изменении. Этот вид связи удобен, если Вам необходимо взаимодействовать с какой–либо базой данных, содержимое которой может динамически обновляться, как например база данных “товары на складе” для больших магазинов — доступ к этой базе одновременно могут иметь разные отделы магазина и список товаров, имеющихся на сладе, постоянно изменяется. Использование горячей связи в этом случае позволит создать локальные списки на компьютерах разных отделов и автоматически обновлять их, как только осуществляется изменение общей базы данных.



WM\_DDE\_ADVISE hWnd aItem & hOptions (Packed Win32)

Младший компонент параметра lParam содержит хендл глобального разделяемого блока, содержащего небольшую структуру DDEADVISE, а старший компонент — атом, описывающий данные на которые осуществляется подписка. Структура DDEADVISE определяет тип связи — горячая или холодная, а также необходимость требовать подтверждения о доставке данных от клиента. В случае платформы Win32 параметр lParam используется для передачи “упакованного” значения. Сервер, получивший это сообщение обязан ответить на него посылкой подтверждения WM\_DDE\_ACK, положительного или отрицательного. Освобождение ресурсов: Клиент, пославший запрос, должен дождаться ответа от сервера. Если ответ положительный, то блок данных должен быть удален сервером, а если отрицательный, то клиент должен сам его удалить.

Структура DDEADVISE содержит следующие данные:

typedef struct { WORD reserved:14, fDeferUpd:1, // 1-”теплая” связь, 0-”горячая” fAckReq:1; // 1-необходимо требовать подтверждение, 0-не надо short cfFormat; // желаемый формат данных } DDEADVISE;

При задании бита fDeferUpd равным 1 осуществляется так называемая теплая связь, при которой сервер только информирует о наличии у него измененных данных, но не посылает их клиенту непосредственно. В случае горячей связи этот бит должет быть равен 0. Бит fAckReq, равный 1, указывает на необходимость требовать от клиента подтверждения о получении данных. Позже, когда сервер начнет посылать клиенту сообщения WM\_DDE\_DATA он установит в структуре DDEDATA бит fAckReq соответствующим тому, который Вы указали в структуре DDEACK и, соответственно, клиент должен будет подтверждать получение им данных от сервера (надо учесть, что бит fAckReq влияет на стратегию освобождения посылаемых данных — подробнее см. описание сообщения WM\_DDE\_DATA).

Поле cfFormat указывает на формат данных, которые клиент ожидает от сервера. Вы можете для одного вида данных в пределах одного DDE–разговора подписаться на получение данных разного типа одновременно, посылая WM\_DDE\_ADVISE столько раз, сколько форматов данных Вы хотите получать. Однако такая возможность разрешена только для горячей связи. Согласно документации подписка на одни и те–же данные в разных форматах для теплой связи запрещена.

Сервер, при получении сообщения WM\_DDE\_ADVISE, проверяет возможность осуществить подписку на эти данные, создает необходимые ему структуры данных и отвечает клиенту сообщением WM\_DDE\_ACK — положительным или отрицательным. Это сообщение в данном случае используется так–же, как и при ответе на WM\_DDE\_DATA.

Если подписка осуществлена успешно, то далее сервер начинает извещать клиента об обновлении данных, осуществляя каждый раз пересылку этих данных клиенту с помощью уже рассмотренного сообщения WM\_DDE\_DATA. Единственное отличие от получения данных в ответ на WM\_DDE\_REQUEST заключается в том, что бит fResponse структуры DDEDATA равен 0, тогда как при отправке данных по запросу он устанавливается в 1. Это позволяет определить причину получения данных при обработке сообщения.



Заканчивается такой обмен посылкой WM\_DDE\_UNADVISE серверу. В этом сообщении указывается, от подписки на какие данные клиент отказывается (так как теоретически один клиент может подписаться сразу на несколько видов данных).



WM\_DDE\_UNADVISE hWnd aItem & cfFormat

Младшее слово параметра lParam описывает формат данных, а старшее слово — атом, описывающий данные на которые осуществляется подписка. В случае платформы Win32 параметр lParam используется также, как и в случае Windows 3.x. Если параметр cfFormat равен 0, то отменяется подписка на указанные данные для всех тех форматов, для которых была осуществлена подписка. Если параметр aItem равен NULL, то отменяется подписка на все виды данных в пределах данного DDE–разговора. Сервер, обработав это сообщение, должен послать в ответ подтверждение WM\_DDE\_ACK, положительное или отрицательное; при этом если одна операция отменяет подписку на данные в разных форматах, то посылается только одно подтверждение.

Сообщение WM\_DDE\_UNADVISE не эквивалентено WM\_DDE\_TERMINATE, после его обработки отменяется только подписка, а сам DDE–разговор продолжается, так что клиент может установить затем новый вид связи с сервером. Сервер обязан ответить на сообщение WM\_DDE\_ADVISE положительным или отрицательным подтверждением.

#### “Теплая” связь — warm link

Протокол при теплой связи. После инициации обмена клиент посылает запрос WM\_DDE\_ADVISE, указывая желаемые данные. Сервер отвечает о возможности теплой связи, и, обычно, сразу же посылает WM\_DDE\_DATA, извещая клиента о возможности немедленного получения данных. Позже сервер посылает WM\_DDE\_DATA клиенту, когда данные обновляются. Сообщения WM\_DDE\_DATA в данном случае не передают самих данных, а только извещают клиента о их наличии у сервера. Протоколы связи и используемые сообщения при теплой связи практически полностью совпадают с протоколами и сообщениями горячи связи, поэтому мы просто рассмотрим те немногие отличия, которые есть между двумя этими видами связи.

Как и горячая связь, теплая устанавливается с помощью сообщения WM\_DDE\_ADVISE с единственным исключением — бит fDeferUpd структуры DDEADVISE равен 1, а не 0.



Далее сервер начинает информировать клиента об обновлении (или наличии) у него нужных данных. Для этого используется обычное сообщение WM\_DDE\_DATA, которое в данном случае не передает никаких данных. Для этого параметр hData (младшее слово lParam или, в случае Win32, младшая компонента, извлекаемая с помощью UnpackDDElParam) устанавливается равным 0. Клиент, получая такое сообщение без данных, может немедленно запросить нужные данные или просто запомнить, что данные изменились и позже, при необходимости, получить их.



Заметьте, что в случае горячей связи вы можете подписаться сразу на получение информации об изменении одних и тех–же данных в разных форматах, а в случае теплой связи это запрещено. Такой запрет объясняется тем, что при поддержке таких видов связи для разных данных или одних и тех–же данных, но в разных форматах, клиент должен вести список данных и форматов на которые осуществлена подписка. При получении WM\_DDE\_DATA, информирующего об изменении данных имя данных задается параметром aItem (упакован в lParam), а формат данных — полем cfFormat структуры DDEDATA. Однако в случае теплой связи сообщение WM\_DDE\_DATA не содержит данных и, следовательно, информация о формате данных недоступна, при этом однозначно найти запись в спике не представляется возможным, если только не наложить ограничение — при теплой связи подписываться можно на данные только в одном формате, тогда запись списка будет однознано определена именем данных.

Для того, что бы получить нужные данные, клиент просто посылает отдельный запрос WM\_DDE\_REQUEST, как в случае холодной связи. Таким образом теплая связь является “гибридом” двух других видов связи — холодной и горячей.



Отмена подписки в случае теплой связи ничем не отличается от такого–же действия для горячей связи — клиент посылает сообщение WM\_DDE\_UNADVISE серверу, который отвечает на это подтверждением.



Работа с DDE в случае теплой или горячей связи обычно не вызывает значительных сложностей; однако надо обратить внимание на возможность использования нескольких видов связи одновременно. Обычно, при написании приложений используются предположения типа “В ответ на посланное WM\_DDE\_REQUEST должно прийти либо WM\_DDE\_DATA, либо WM\_DDE\_ACK”. Однако в жизни ситуация может оказаться существенно сложнее — если одно окно использует несколько соединений, то в ответ на посланный WM\_DDE\_REQUEST сначала может прийти сообщение WM\_DDE\_DATA, информирующее о получении данных (в случае теплой или горячей связи) и только потом настоящий ответ на WM\_DDE\_REQUEST. Для того, что бы уменьшить вероятность таких ситуаций, для каждого DDE–разговора создают специальное скрытое окно, взаимодействующее с сервером (или клиентом). Однако при разработке собственных приложений такую возможность все–равно надо учитывать, так как в процедурах обработки сообщений придется проверять, в связи с каким событием получено то или иное сообщение; а при разработке DDE–клиентов надо постараться разделить разные виды обмена данными либо по разным DDE–разговорам, либо по времени.

#### Передача данных от клиента к серверу

При обмене данными между клиентом и сервером возможно не только получение клиентом данных, содержащихся на сервере, но и возможность передачи некоторых данных от клиента к серверу. Для этого предусмотрено специальное сообщение WM\_DDE\_POKE, которое посылается клиентом серверу для записи данных.



WM\_DDE\_POKE hWnd aItem & hData (Packed Win32)

Сообщение WM\_DDE\_POKE посылается клиентом для передачи данных серверу. В случае платформы Windows 3.x младший компонент параметра lParam содержит хендл передаваемого блока данных, а старший — атом имени передаваемых данных. Передаваемый блок данных должен быть глобальным и разделяемым и должен начинаться со структуры DDEPOKE. При посылке сообщения WM\_DDE\_POKE используется следующая стратегия освобождения блока данных: он уничтожается пославшим приложением (клиентом), если:

* в заголовке блока бит fRelease равен 0
* клиент вернул отрицательный ответ
* посылка данных не состоялась

Получившее данные приложение (сервер) будет уничтожать этот блок только если бит fRelease равен 1 и сервер возвращает положительное подтверждение.

typedef struct { WORD unused:13, fRelease:1, // 1 - блок освобождается клиентом fReserved:2; short cfFormat; // формат данных (см. форматы буфера обмена) BYTE Value[1]; // передаваемые данные } DDEPOKE;

Сервер, получивший это сообщение должен сохранить полученные данные и сообщить клиенту о результате выполнения этой операции посылкой сообщения WM\_DDE\_ACK, положительного или отрицательного. В этом сообщении может быть передан полученный атом имени данных, либо полученный может быть удален, а для посылки подтверждения создан заново.

### Выполнение команд DDE

Помимо механизма обмена данными между клиентом и сервером DDE предусматривает еще один способ взаимодействия приложений — выполнение сервером команд клиента. Для того, что бы указать серверу на необходимость выполнения команды, клиент должен послать специальное сообщение WM\_DDE\_EXECUTE, в ответ на которое сервер пытается выполнить команду и сообщает клиенту о результате ее выполнения, посылая в ответ сообщение WM\_DDE\_ACK (положительное или отрицательное).



WM\_DDE\_EXECUTE hWnd hCmd & 0 (Windows 3.x)

WM\_DDE\_EXECUTE hWnd hCmd (Win32)

Параметр lParam описывает команду, передаваемую серверу. Для этого строка, содержащая команду, помещается в блок данных, выделяемый с помощью функции GlobalAlloc с флагом GMEM\_DDESHARE. В случае платформы Windows 3.x старшее слово параметра lParam содержит хендл этого блока, а младшее — просто 0; А в случае платформы Win32 параметр lParam непосредственно содержит этот хендл. Освобождение ресурсов: блок данных, содержащий команду, должен быть освобожден клиентом при получении подтверждения от сервера. Вместе с этим подтверждением возвращается хендл этого блока, так что клиент свободно может его освободить.

В случае платформы Win32 строка, содержащая команду может быть представлена UNICODE–строкой, но только в том случае, если оба участвующие в DDE–разговоре окна поддерживают UNICODE, то есть оба возвращают TRUE при вызове функции IsWindowUnicode( hWnd ).

В ответ на сообщение WM\_DDE\_EXECUTE сервер посылает сообщение WM\_DDE\_ACK, информирующее о выполнении команды (положительное) или о неудаче (отрицательное). В этом сообщении возвращается хендл блока, содержащего команду, так что сервер может этот блок удалить, а также в параметре wStatus — в младшем байте — может быть возвращен дополнительный код, уточняющий результат или причину ошибки (см. описание структуры DDEACK, представляющей параметр wStatus в разделе, посвященном холодной связи).

WM\_DDE\_ACK hWnd hCmd & wStatus (Packed Win32)

Младший компонент lParam содержит информацию о посылаемом подтверждении. Младший байт этого слова содержит код, возвращаемый приложением, старший байт содержит два значащих бита 0x80 и 0x40. Часто этот параметр представляется в виде структуры DDEACK. Старший компонент lParam представляет собой хендл блока, содержащего команду. Согласно документации этот блок обязательно должен быть освобожден при обработке этого сообщения. В случае платформы Win32 параметр lParam используется для передачи хендла “упакованных” данных.

При разработке собственных DDE–серверов, исполняющих какие–либо команды, надо обратить внимание на формат записи этих команд. Разработчики Microsoft предполагают совершенно определенный формат записи команд, который, к сожалению, часто не поддерживается сторонними разработчиками. Конечно, с точки зрения совместимости, желательно придерживаться этого формата.

В соответствии с документацией Microsoft команда должна представлять собой строку, оканчивающуюся нулевым символом и содержащую одну или несколько команд. Каждая команда должна быть заключена в квадратные скобки […]. Каждая команда состоит из двух частей — имени команды и, возможно, списка параметров. При этом список параметров заключается в круглые скобки, а параметры в списке разделяются запятыми. Имя команды не может содержать пробелов, скобок, запятых и кавычек; параметры могут содержать эти символы, но в таком случае такой параметр должен быть заключен в кавычки, а символ кавычек в тексте параметра представляется как повторяющаяся кавычка. В прежних версиях DDE — для Windows 3.x — требовалось, что бы символы ( ) [ ], встречаемые в тексте параметра (в кавычках) дублировались; в то время как в Win32 это уже не требуется. Проблема, однако, существует — в среде Win32 могут быть запущены как 16–ти разрядные приложения, так и 32–х разрядные, при этом возможно взаимодействие таких приложений посредством DDE. В такой ситуации серверы должны распознавать оба варианта представления скобок — как повторяющиеся, так и не повторящиеся.

Примеры для Win32:

[команда] [команда1][команда2][команда\_с\_параметрами(параметр1,параметр2)][командаN] [команда(параметр1,”параметр2 с пробелами, скобками []() и “” кавычками”)]

В случае Windows 3.x последний пример будет выглядеть так:

[команда(параметр1,”параметр2 с пробелами, скобками [[]](()) и “” кавычками”)]

Если строка содержит несколько заключенных в квадратные скобки команд, то сервер должен разобрать эту строку по командам и обработать их последовательно. Вариант с передачей нескольких команд в одной строке имеет одно неудобство — возвращается результат выполнения всей последовательности команд, причем понять где возникла ошибка (скажем, синтаксис команды или недостаток свободных ресурсов) будет практически невозможно.

### Тема DDE–разговора “System”

Согласно документации Microsoft всем вновь разрабатываемым DDE–серверам рекомендуется поддерживать в каждом сервисе специальную служебную тему DDE–разговора “System”. Эта тема предназначена для получения основной информации о сервере. В пределах этой темы определено несколько стандартных имен данных, к которым можно обратиться для получения требуемой информации.

К сожалению сами разработчики Microsoft не очень строго соблюдают это правило. Так, например, Microsoft Word и Microsoft Excel поддерживают эту тему, а Program Manager, Explorer, Internet Explorer — не поддерживают. Однако документация остается документацией, поэтому при разработке собственных DDE–серверов стоит придерживаться предлагаемых правил.

При обмене данными с серверами, поддерживающими тему “System” надо придерживаться следующих правил:

* для получения данных используется холодная связь
* данные предоставляются только в формате CF\_TEXT
* по многим запрашиваемым данным возвращается список значений. Этот список представлен в виде строки, оканчивающейся нулевым символом, отдельные пункты которой отделяются символом табуляции (код — 9).
* При использовании DDEML имя темы “System” определено в файле DDEML.H как SZDDESYS\_TOPIC.

Помимо темы “System” предусмотрен специальное название данных, которые должны поддерживаться во всех остальных темах, с помощью которого можно получить информацию о всех поддерживаемых именах данных в данной теме.

При разработке собственного сервера не обязательно поддерживать все приведенные данные, равно как не обязательно вообще поддерживать тему “System” (хотя это желательно). Если Вы поддерживаете только некоторые данные этой темы, обратите внимание на данные “SysItems”, которые позволяют клиенту узнать, какие именно данные можно запрашивать.

|  |  |
| --- | --- |
| Название темы для DDE Название темы для DDEML | Описание |
| Formats SZDDESYS\_ITEM\_FORMATS | Возвращает список поддерживаемых сервером форматов данных. Обычно названия совпадают с названиями форматов буфера обмена, но первые три символа “CF\_” опущены. То есть формат CF\_BITMAP будет представлен как BITMAP. Рекомендуется так упорядочивать названия форматов в списке, что бы форматы передающие больше информации были первыми. То есть, например, формат DIB должен быть расположен до формата BITMAP. |
| Help SZDDESYS\_ITEM\_HELP | Пояснения к использованию данного DDE сервера. Достаточно произвольный текст, желательно поясняющий, какие данные могут быть получены, какие команды могут быть выполнены какие данные могут быть переданы серверу и т.д. |
| ReturnMessage SZDDESYS\_ITEM\_RTNMSG | Информация о последнем отправленом подтверждении WM\_DDE\_ACK. С помощью этого типа данных можно передавать дополнительные данные, уточняющие результат выполнения последней транзакции. |
| Status SZDDESYS\_ITEM\_STATUS | В ответ на этот запрос сервер должен отправить строку “Busy” или “Ready”, информирующую о состоянии сервера. Заметьте, что эту строку сервер должен посылать, даже если он занят и не может обрабатывать иные запросы. |
| SysItems SZDDESYS\_ITEM\_SYSITEMS | Возвращает список имен данных, которые представлены в теме “System”. Этот список может использоваться клиентом, что бы узнать, какие данные он может запрашивать в рамках темы “System”. |
| Topics SZDDESYS\_ITEM\_TOPICS | Возвращает список тем, поддерживаемых сервером в данный момент времени. В процессе работы сервера этот список может изменяться, так, например, Microsoft Word открывает отдельную тему для работы с каждым загруженным документом. |
| TopicItemList SZDDE\_ITEM\_ITEMLIST | Аналогично SysItems, возвращает список данных, поддерживаемых для данной темы. Этот запрос может применяться к темам, не являющимися темой “System”. Этот список может изменяться в процессе работы сервера. |

### Program Manager в качестве DDE–сервера

Program Manager является специализированным DDE–сервером. Он позволяет запущенным приложениям управлять конфигурацией групп приложений, изменять их атрибуты и т.д. Обычно программы установки приложений используют DDE с Program Manager для создания необходимой группы и наполнения ее элементами. В случае Windows–94 или Windows NT v4.0 и выше Program Manager обычно не используется, но работающий при этом Explorer поддерживает те–же самые сервис, тему и команды, что и Program Manager. При этом вместо групп создаются подменю в меню Пуск|Программы (Start|Programs).

В принципе Windows может быть сконфигурирован так, что вместо Program Manager используется совершенно иная оболочка, которая может не поддерживать описываемый DDE. Возможным выходом из этой ситуации является попытка запуска Program Manager (PROGMAN.EXE), если с первого раза не удается установить DDE–разговор.

Для взаимодействия с Program Manager необходимо установить DDE–разговор с сервером, поддерживающим сервис и тему с одинаковым именем “PROGMAN”, а затем можно передавать необходимые команды (сообщение WM\_DDE\_EXECUTE). Команды, которые поддерживает Program Manager, позволяют приложению создавать, отображать, удалять и перезагружать группы, добавлять, изменять или удалять элементы групп и завершать работу Program Manager. Для этого предназначены следующие команды:

CreateGroup — создать группу программ

ShowGroup — показать группу в указанном состоянии

Reload — перезагрузить группу из файла

DeleteGroup — удалить группу

AddItem — добавить элемент

ReplaceItem — изменить элемент

DeleteItem — удалить элемент

ExitProgman — выйти из Program Manager

Program Manager обрабатывает команды в формате, рекомендованном Microsoft. То есть каждая команда заключается в квадратные скобки, а если она имеет дополнительные параметры, то список параметров заключается в круглые скобки, причем параметры в списке разделены запятыми. В одном запросе к серверу можно указать несколько команд сразу, например:

[ShowGroup(“Accessories”,1)][AddItem(myapp.exe,”My app”,myapp.exe,5)]

Этот пример добавляет элемент “My app” в группу “Accessories”.

Помимо выполнения этих команд Program Manager может выполнять обмен данными с клиентом, благодаря чему возможно получение информации об имеющихся группах и элементах этих групп. Для этого используются те же имена сервиса и темы “PROGMAN”, что и для выполнения команд, но при этом серверу передаются запросы на получение данных по холодной связи в формате CF\_TEXT.

Для того, что бы получить список групп, клиент должен прочитать данные с именем “Group”. В ответ Program Manager вернет список групп, разделенный символами перевода строки.

Приложение может запросить у Program Manager информацию о группе. Для этого оно должно прочитать данные с именем, соответствующем имени группы. Возвращается текст из нескольких строк, каждая строка состоит из нескольких компонент, разделяемых запятыми. Первая строка определяет информацию о самой группе, а все последующие — об элементах группы.

Информация о группе состоит из:

* имени группы, заключенного в кавычки
* пути к файлу группы (.grp)
* число элементов в группе

Информация об элементе группы:

* команда, заключенная в кавычки
* название каталога
* пути к файлу, содержащему пиктограмму
* индексу пиктограммы в файле
* “горячей клавише” (в числовой форме)
* флагу минимизированного состояния при запуске приложения

Примечание: в документации утверждается, что на количество элементов в группе наложено ограничение — не более 50 элементов на одну группу. Однако существует иное, гораздо более жесткое ограничение — размер grp–файла под Windows 3.x не может быть больше 64K. Этот размер может быть легко превышен, если используются видео–режимы с большим количеством цветов (Hi–Color или TrueColor), например 16, 24 или 32 бита на пиксел. Дело в том, что grp–файл содержит в себе изображения всех пиктограмм, размер которых увеличивается с увеличением числа цветов. При этом предельный размер grp–файла может быть достигнут после десятка элементов (для TrueColor — примерно 13 элементов).

Далее более подробно описываются команды, обрабатываемые Program Manager. При этом мы будем придерживаться следующего синтаксиса — описание самой команды в квадратные скобки не заключается, а необязательные параметры выделяются с их помощью. Для лучшей читаемости параметры разделяются дополнительным пробелом (при передачи команды этих пробелов быть не должно).

Рекомендуется дополнительно просмотреть раздел “Выполнение команд DDE” для получения справок о синтаксисе записи команд и параметров.

CreateGroup( GroupName [,CommonGroupFlag] ) CreateGroup( GroupName [,GroupFile] )

Команда CreateGroup создает указанную группу, либо делает активной уже существующую.

Параметры:

GroupName строка, задающая имя группы.

CommonGroupFlag параметр указывает, какая группа создается. Если он равен 1, то создается общая группа, а если 0, то персональная. Если работающий пользователь не имеет административных привилегий, то попытка создать общую группу не удастся. По умолчанию используется 1 для пользователей, располагающих привилегиями администратора, и 0 для остальных пользователей.

GroupFile в Windows 3.1 этот параметр задавал имя grp–файла. В старших версиях Windows в качестве второго параметра рассматриваются только 0 и 1, иначе этот параметр игнорируется.

ShowGroup( GroupName, ShowCommand [,CommonGroupFlag] )

Команда ShowGroup позволяет показать группу в нормальном, максимизированном или минимизированном виде.

Параметры:

GroupName строка, задающая имя группы.

ShowCommand целое число, указывающее, в каком виде надо отобразить окно группы.

1 — отобразить активным в нормальном состоянии

2 — отобразить активным в минимизированном состоянии

3 — отобразить активным в максимизированном состоянии

4 — отобразить окно группы в нормальном состоянии; активное окно останется активным

5 — делает окно группы активным, не изменяя ее размера и положения

6 — минимизирует окно группы

7 — минимизирует окно группы; активное окно останется активным

8 — отображает в текущем состоянии

CommonGroupFlag параметр указывает, какая группа создается. Если он равен 1, то создается общая группа, а если 0, то персональная. Если работающий пользователь не имеет административных привилегий, то попытка создать общую группу не удастся. По умолчанию используется 1 для пользователей, располагающих привилегиями администратора, и 0 для остальных пользователей.

Reload( GroupName [,CommonGroupFlag] )

Эта команда заставляет Program Manager удалить описание группы из памяти и загрузить grp–файл снова. Такая возможность может применяться, если программа модифицирует непосредственно grp–файл, а затем хочет отобразить сделанные изменения.

Параметры:

GroupName строка, задающая имя группы.

CommonGroupFlag параметр указывает, какая группа создается. Если он равен 1, то создается общая группа, а если 0, то персональная. Если работающий пользователь не имеет административных привилегий, то попытка создать общую группу не удастся. По умолчанию используется 1 для пользователей, располагающих привилегиями администратора, и 0 для остальных пользователей.

DeleteGroup( GroupName [,CommonGroupFlag] ) DeleteGroup( GroupName [,GroupFile] )

Команда DeleteGroup удаляет указанную группу.

Параметры:

GroupName строка, задающая имя группы.

CommonGroupFlag параметр указывает, какая группа создается. Если он равен 1, то создается общая группа, а если 0, то персональная. Если работающий пользователь не имеет административных привилегий, то попытка создать общую группу не удастся. По умолчанию используется 1 для пользователей, располагающих привилегиями администратора, и 0 для остальных пользователей.

GroupFile в Windows 3.1 этот параметр задавал имя grp–файла. В старших версиях Windows в качестве второго параметра рассматриваются только 0 и 1, иначе этот параметр игнорируется.

AddItem( CmdLine [,Name [,IconPath [,IconIndex [,xPos, yPos [,DefDir [,HotKey [,fMinimize [fSeparateMemSpace] ] ] ] ] ] ] )

Команда AddItem позволяет добавить в активную группу программный элемент.

Параметры:

CmdLine строка, задающая команду, осуществляющую запуск приложения. Эта строка должна содержать имя приложения, а также может содержать полный путь к приложению и другие необходимые параметры.

Name задает имя элемента в группе. Если этот параметр опущен, то используется имя приложения.

IconPath задает полное имя файла, содержащего ресурс пиктограммы, которая будет отображаться в окне группы. Это может быть выполняемый файл Windows (.exe, .dll) или отдельная пиктограмма (.ico). Если этот параметр опущен, то пиктограмма ищется в файле приложения, причем, если указанный файл не является приложеним, то используется пиктограмма ассоциированного с данным типом файлов приложения (ассоциации задаются с помощью реестра Windows); а если и это не удается, то используется пиктограмма по умолчанию.

IconIndex задает индекс пиктограммы в файле, определенном параметром IconPath.

xPos, yPos задают положение пиктограммы в окне группы. Эти параметры представлены целыми числами. Если позиция не задана, то Program Manager находит первое неиспользуемое место.

DefDir задает имя рабочего каталога приложения.

HotKey задает “горячую” клавишу, используемую для вызова приложения.

fMinimize указывает, надо–ли отображать окно приложения при запуске в виде пиктограммы.

fSeparateMemSpace указывает, надо–ли запускать 16–ти разрядное приложение в самостоятельном адресном пространстве (используется на некоторых платформах Win32).

ReplaceItem( ItemName )

Команда ReplaceItem используется для замены одного элемента другим. При выполнении этой команды указанный элемент удаляется, а его позиция запоминается. Последующий вызов AddItem создаст новый элемент в этом месте.

Параметры:

ItemName строка, задающая имя удаляемого элемента. На его месте будет размещен следующий создаваемый элемент.

DeleteItem( ItemName )

Команда DeleteItem осуществляет удаление указанного программного элемента.

Параметры:

ItemName строка, задающая имя удаляемого элемента.

ExitProgman( bSaveGroups )

Эта команда завершает работу Program Manager.

Параметры:

bSaveGroups величина, указывающая надо или нет сохранять текущее состояние групп перед завершением работы Program Manager. При значении 0 сохранения групп не происходит.