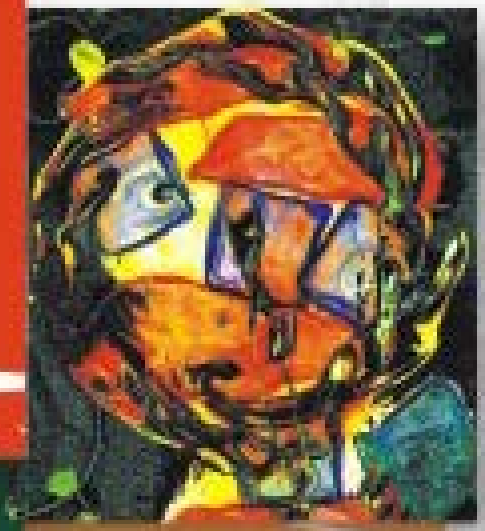
ДМИТРИЙ КУЗАН
ВЛАДИМИР ШАПОРОВ

bhv

ПРОГРАММИРОВАНИЕ Win32 API В DELPHI



- ВВЕДЕНИЕ В Win32 API
- GDI+ – ИНТЕРФЕЙС НОВОГО ПОКОЛЕНИЯ
- SIMPLE MAPI – РАБОТА С ЭЛЕКТРОННОЙ ПОЧТОЙ
- VIDEO FOR WINDOWS – РАБОТА С ВИДЕО
- MEDIA CONTROL INTERFACE – РАБОТА С МУЛЬТИМЕДИА
- TAPI – РАБОТА СО СРЕДСТВАМИ КОММУНИКАЦИЙ

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

УДК 681.3.06
ББК 32.973.26-018.2
К89

Кузан Д. Я., Шапоров В. Н.

К89 Программирование Win32 API в Delphi. — СПб.: БХВ-Петербург, 2005. — 368 с.: ил.

ISBN 5-94157-535-1

Рассмотрено применение различных интерфейсов прикладного программирования Windows (Win32 API) при разработке приложений с использованием Borland Delphi. Описаны основы работы с API. Подробно освещены вопросы практического применения API при создании приложений для работы с электронной почтой (MAPI), со средствами коммуникаций (TAPI), мультимедиа (MMCI), графическим интерфейсом и др. Материал сопровождается наглядными практическими примерами. На компакт-диске расположены исходные тексты примеров, программы и необходимые библиотеки.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Татьяна Лапина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.09.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 29,67.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-535-1

ы

Оглавление

ГЛАВА 2

TAPI – интерфейс программирования приложений для работы с телефонией

Введение в TAPI

Бурное развитие телефонных коммуникаций, а также повсеместное использование компьютеров в сферах связи сделало большой прорыв в деле передачи информации по линиям связи. И хотя широко шагающий прогресс преподносит нам все новые и новые виды коммуникаций — от оптоволокна до спутниковой связи — телефон по-прежнему остается самым распространенным средством связи, а наличие модема вкупе с компьютером открывает широкий доступ, начиная от простой передачи факсов и файлов по телефонной линии до выхода в Интернет. В данной главе мы постараемся рассмотреть интерфейс прикладного программирования, специально созданный для управления телефонией для Windows. Имя этому интерфейсу — TAPI.

TAPI — Telephony Application Programming Interface или интерфейс программирования и работы с телефонией в операционной системе Windows. Используя данный интерфейс, вы можете создавать прикладные приложения, поддерживающие широкий диапазон телефонных коммуникаций и разнообразных услуг, использующих телефонную линию. В ваши руки попадет возможность использовать обычную телефонную линию как для передачи данных, так и для передачи речи. Данный интерфейс является неотъемлемой частью операционной системы Microsoft Windows и представляет из себя открытую архитектуру, созданную для упрощения программирования коммуникационных приложений. Высокоуровневые процедуры и функции TAPI скрывают сложности программирования коммуникаций, на низком уровне, позволяя разработчику отбросить в стороны все проблемы, связанные с управлением конкретным коммуникационным устройством, в частности модемом. Однако это не значит, что в TAPI отсутствуют средства низкоуровне-

вого программирования. Также следует отметить, что TAPI позволяет создавать коммуникационные приложения, независимые от установленных устройств коммуникаций, и представляет следующие возможности:

- автоматический набор номера;
- посылка и получение файлов, факсов и электронной почты;
- работа с каналами новостей и информационных услуг;
- возможность управления голосовой почтой;
- автоматизация и обработка поступающих вызовов.

Прежде чем приступить к непосредственному разбору TAPI, мы немного остановимся на взаимодействии Дельфийских приложений с TAPI. Для подключения TAPI в Delphi вы должны поместить в секцию Uses ссылку на файл TAPI.pas, являющийся заголовочным файлом динамической библиотеки TAPI.DLL.

Заголовочный файл TAPI.pas вы можете взять с прилагаемого диска из каталога Install\TAPI Headers\TAPI\.

> ***Примечание***
>
> Каталог Install\TAPI Headers\JEDI Headers Translations\ содержит заголовочные файлы проекта JEDI. В наших примерах данные заголовочные файлы не используются, однако они могут кому-либо понадобится, так что мы их включили в данный диск на всякий случай.

При использовании функций TAPI.pas ваше приложение обращается к TAPI.DLL, которая, в свою очередь, переназначает вызов к TAPI32.DLL. Далее вызов идет к сервису поставщика телефонных услуг (Telephony Service Provider Interface) TAPISRV.EXE, который связывается с конкретными низкоуровневыми драйверами коммуникационных устройств, выполняющими управление, например, HAYES-совместимыми модемами.

Интерфейсы и уровни программирования TAPI

В TAPI реализовано два интерфейса программирования — высокоуровневое и низкоуровневое. Данные интерфейсы предоставляют отдельные функции и структуры для управления коммуникациями. В то время как высокоуровневый интерфейс предлагает горстку функций, упрощающих работу программиста, низкоуровневый интерфейс обеспечивает функциональные дополнительные возможности, а также представляет программисту необходимую гибкость работы с TAPI.

Помимо этого интерфейсы разделены на несколько уровней:

- базовый (Basic Telephony);
- вспомогательный (Supplementary Telephony);
- расширенный (Extended Telephony).

Уровень интерфейсов показывает, насколько тесно приложение, вызывающее функцию, взаимодействует с TAPI.

Стоит заметить, что функции подразделены также на синхронные и асинхронные. Разница между ними состоит в способе обмена данными с коммуникационным устройством. Синхронные функции всегда ожидают ответа на свой запрос, приостанавливая работу приложения; асинхронные ждут ответа, не приостанавливая работу.

После краткого введения мы можем более подробно остановиться на уровнях интерфейсов. Их, как вы знаете, три.

Базовый уровень

На данном уровне находятся функции по спецификации TAPI, которые должны быть совместимы со всеми устройствами коммуникаций, т. е. все устройства коммуникаций, работающие с TAPI, должны полностью поддерживать "базовый" набор функций. Функции этого уровня работают с сервисом POTS.

Примечание

POTS (Plain Old Telephone Service) — базовый телефонный сервис, основанный на стандартных одноканальных (single-line) телефонах и телефонных линиях.

Стандартную одноканальную схему вы можете увидеть на рис. 2.1.

Рис. 2.1. Стандартное одноканальное соединение

Вспомогательный уровень

На вспомогательном уровне находятся "базовые" функции, расширенные дополнительными параметрами или флагами. Расширение функций базового уровня связано с тем, что на данном уровне находятся функции, поддерживающие современные коммуникационные технологии и не вписывающиеся в базовый сервис POTS.

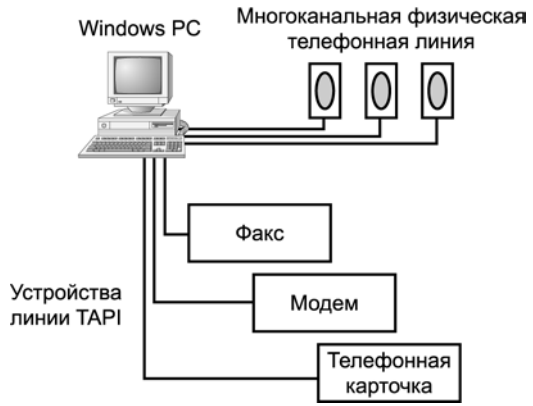


Рис. 2.2. Многоканальные соединения

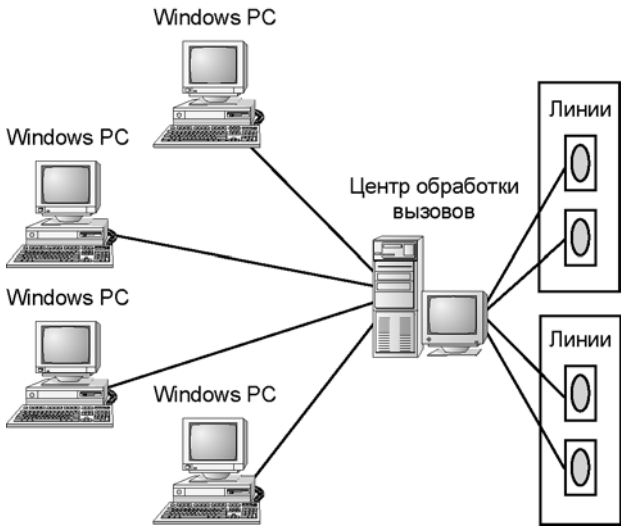


Рис. 2.3. Центр обработки вызовов

Например, на вспомогательном уровне предусмотрена поддержка конференц-связи, многоканальных соединений и офисной телефонной сети (Public

Branch Exchange, PBX). Так, например, структурную схему многоканального соединения вы можете увидеть на рис. 2.2, а на рис. 2.3 — структурную схему центра обработки вызовов. В роли центра обработки может выступать как офисная АТС, так и специализированный компьютер.

Расширенный уровень

В расширенный уровень входят функции, которые являются зависимыми от конкретного устройства коммуникаций. Используя функции из расширенного набора, следует иметь в виду, что ваше приложение становится зависимым от конкретного оборудования. Так как расширенный уровень включает в себя и вспомогательный уровень, это позволяет расширить возможности программы за счет более тесного взаимодействия с конкретным поставщиком услуг или конкретным коммуникационным устройством.

Работа с устройствами линий

Большая часть функций, констант и структур TAPI — это часть API, работающая с так называемыми "устройствами линий". По мере выхода все новых версий TAPI данная часть API увеличивается. В данном разделе мы расскажем об основных операциях с "устройствами линий", такими как инициализация, открытие, закрытие, настройка, а также операциях получения различной статусной информации. Прежде всего, мы рассмотрим понятие "устройства линии". Документация по TAPI трактует устройство линии (или просто "линия") как коммуникационное устройство, связанное с телефоном, т. е. с физической линией. Коммуникационное устройство может быть факсом, модемом или ISDN-картой. Устройства линии обеспечивают широкий диапазон функциональных возможностей телефонии, позволяя вашему приложению посылать или получать разнообразную информацию по телефонной сети.

Следует рассматривать устройства линии не как физическую телефонную линию, а как, скорее всего, логическое представление аппаратных коммуникационных средств вкупе с физической линией связи.

Основные шаги работы с телефонией

Рассматривая работу с телефонией, стоит обратить внимание на то, что последовательность действий очень похожа на работу с другими API, например, с мультимедией. И в том и в другом случае сначала вы должны установить связь с устройством путем его инициализации.

Первичная инициализация TAPI выполняется с помощью функций `lineInitialize` или `lineInitializeEx`. Стоит заметить, что функция

lineInitialize является устаревшей и оставлена для совместимости с версией TAPI 1.4. Вместо нее рекомендуется использовать функцию lineInitializeEx. Данные функции имеют пять параметров, которые хранят информацию относительно TAPI-сессии, включая адрес функции отзыва (Callback), которая обращается с сообщениями Windows. Новая функция включает два дополнительных параметра, указывающие на структуру LINEINITIALIZEEXPARAMS, которая содержит дополнительные параметры, используемые для связи между вашим приложением и TAPI.

После первичной инициализации TAPI и установки связи вы должны предпринять следующие шаги:

1. Проверить, сможет ли TAPI-устройство выполнять заданные вами функции TAPI (если, конечно, данные функции не входят в базовый набор). Данная проверка называется проверкой способностей телефонии (TAPI). Так как не все функции могут обрабатываться конкретным коммуникационным устройством, перед непосредственным выполнением их следует проверять на совместимость. Например, не любой модем может обрабатывать звуковые команды Voice-модема, а только Voice-модем. Для того чтобы проверить способность устройства TAPI, вы должны вызывать следующие функции: lineGetDevCaps и lineGetAddressCaps.
2. Открыть идентифицированное вами коммуникационное устройство и настроить механизмы сообщений TAPI на ваше приложение. Механизмы сообщений TAPI сродни сообщениям Windows и применяются для информирования приложения обо всех изменениях в линии и в работе коммуникационного устройства. К примеру, определенное сообщение TAPI может информировать вас об обрыве линии.
3. Выполняете определенную работу и закрываете коммуникационное устройство и TAPI.

Далее мы постараемся подробно разобрать каждый шаг.

Конфигурирование и настройка устройства коммуникации

Очень часто пользователям требуется возможность вручную редактировать конфигурацию устройства коммуникации из программы. TAPI представляет программисту для этого две функции — lineConfigDialog и lineConfigDialogEdit. Обе функции вызывают стандартное диалоговое окно настройки и конфигурирования коммуникационного устройства. Пользователь сам может настроить параметры, связанные с данным устройством. Хоть эти функции и схожи по принципу работы, у них есть существенное разли-

чие: функция lineConfigDialog полностью заменяет конфигурацию оборудования, что может быть чревато последствиями (при неправильном конфигурировании устройства неквалифицированным пользователем можно расстроить работу устройства). Функция lineConfigDialogEdit более демократична, т. к. записывает информацию как бы в буфер (структура VarString) и не вносит изменения сразу. Вы можете позже вызвать функцию lineSetDevConfig и модернизировать конфигурацию на основании записанных в буфер данных.

Также функции позволяют показать любое подокно настройки конфигурации основного диалогового окна. Достигается это использованием параметра lpszDeviceClass. Соответствующие строковые константы окон для lpszDeviceClass вы можете найти в файле помощи TAPI.

Структура *VarString TAPI*

Как упоминалось ранее, функция LineConfigDialogEdit записывает информацию о конфигурации и настройках оборудования в структуру хранения конфигурации. Данная структура VarString определена в TAPI следующем образом (описание структуры взято с перевода TAPI проектом JEDI):

```
PVarString = ^TVarString;
varstring_tag = packed record
dwTotalSize, dwNeededSize,
dwUsedSize, dwStringFormat,
dwStringSize, dwStringOffset: DWORD;
// Modified by Alan C. Moore: new field, next line added
data : array [0..0] of byte;
end;
```

Обратите внимание, что в файле перевода заголовка TAPI от проекта JEDI присутствует дополнительное поле data, где хранятся дополнительные данные об коммуникационном устройстве.

Первые три поля этой структуры — dwTotalSize, dwNeededSize и dwUsedSize — являются обычными полями информации о структуре и часто встречаются в других API Microsoft. Поле dwTotalSize указывает полный размер (в байтах) структуры данных. Вся ответственность по выделению памяти для структур ложится, как мы говорили, на программиста, т. к. в разных версиях TAPI размер структуры может меняться.

Как работать с переменными данными структуры при выделении памяти? Подход заключается в том, чтобы выделить область памяти как бы с запасом под блок переменных данных. Пример в листинге 2.1 покажет вам, как этого можно добиться.

Листинг 2.1. Работа с переменными данными на примере `VarString`

```
if FDeviceConfig=Nil then
begin
   // Выделяем память с запасом в 1000 байт
   FDeviceConfig := AllocMem(SizeOf(VarString)+1000);
   // Обнуляем область памяти
   FillChar(FDeviceConfig^, SizeOf(VarString)+1000, 0);
   // Устанавливаем размер структуры
   FDeviceConfig.dwTotalSize := SizeOf(VarString)+1000;
   // Устанавливаем тип структуры
   FDeviceConfig.dwStringFormat := STRINGFORMAT_BINARY;
end;
```

Поле `dwNeededSize` содержит информацию о размере (в байтах) возвращенной структурой информации. Поле `dwUsedSize` содержит информацию о размере (в байтах) полезной информации, содержащейся в структуре. Данные поля устанавливаются функциями `lineConfigDialog` и `lineConfigDialogEdit`, которые и заполняют данную структуру информацией о конфигурации.

Перед вызовом функции `lineConfigDialogEdit` вы должны вызвать функцию `lineGetDevConfig` и считать начальные данные конфигурации. Эти данные всегда связаны с конкретным коммуникационным устройством (устройством линии). Для обычного модема пользователь, например, может увидеть такие параметры, как `rate`, `character format`, `modulation schemes` и `error control protocol settings`.

Всякий раз, когда вы открываете линию с константой `LINEMAPPER`, вы должны получить идентификатор линии ID с помощью функции `lineGetID`. Данный ID будет затребован во многих функциях TAPI. В нашем случае он нужен в функции `lineGetDevConfig`.

> ***Замечание***
>
> Поскольку структура `VarString` заполняется определенным устройством, может иметь разный размер для каждого устройства, и данные могут быть разными, вы не можете напрямую манипулировать данными, хранящимися в структуре `VarString`. Запомните, данные структуры `VarString` являются данными для внутреннего использования TAPI. Также вы не можете передавать данные о конфигурации одного устройства другому, даже если эти устройства одного класса.

Поговорим подробно об инициализации TAPI.

Как было сказано, установка первичной инициализации TAPI осуществляется функциями `lineInitializeEx` или `phoneInitializeEx`.

Когда ваше приложение вызывает любую из этих функций, TAPI начинает настраивать среду телефонии. Стоит заметить, что на данном уровне не происходит прямого взаимодействия с коммуникационным устройством TAPI. Настройка среды телефонии включает в себя загрузку TAPI.DLL, TAPISRV.EXE и загрузку соответствующих драйверов коммуникационных устройств, указанных в реестре Windows.

Если TAPI решит, что инициализация невозможна, функции могут вернуть следующие коды ошибок: `INIFILECORRUPT`, `LINEERR_NODRIVER`. Первая ошибка заключается в том, что TAPI не может правильно определить настройки телефонии из реестра, вторая заключается в том, что TAPI не смог загрузить соответствующие драйверы коммуникационных устройств или их дополнительных модулей, зарегистрированных в службе телефонии.

Использование функций `lineInitializeEx` или `phoneInitializeEx` всегда должно идти в комплекте с функцией `lineShutdown`, т. е. всегда должен быть реализован механизм открытия-закрытия. Нельзя второй раз открыть TAPI, пока не будет осуществлено закрытие `lineShutdown`. Если по каким-либо причинам вы забудете закрыть TAPI, система телефонии не сможет корректно освободить занимаемые ресурсы.

После успешного завершения функции `lineInitializeEx` возвратятся два параметра, очень важных для последующей работы с телефонией. Это дескриптор TAPI и идентификатор устройства линии (коммуникационного устройства). Как мы кратко упоминали ранее, функция `lineInitializeEx` возвращает в параметре `dwNumDevs` количество коммуникационных устройств, доступных через TAPI. Идентификационные номера устройств лежат в диапазоне от 0 до `dwNumDevs-1`.

Другой критической особенностью TAPI является контроль версий TAPI и способность телефонии. Как мы упоминали в *разд. "Основные шаги работы с телефонией"*, ваше приложение всегда должно проверять способность TAPI корректно выполнить заданные команды. Имейте в виду, что умение TAPI выполнять определенные команды (функции) зависит от многих факторов, таких как конфигурация сети, наличие определенных аппаратных и программных средств (в особенности драйверов), установленных на компьютере, версии TAPI.

Разберем более подробно данные особенности инициализации.

Три механизма уведомлений (сообщений) TAPI

При использовании TAPI-функции `lineInitializeEx` вы должны выбрать один из трех механизмов получения уведомлений (сообщений). Любой из них позволит вашему приложению получать информацию о событиях телефонии.

TAPI представляет следующие:

- Hidden Window — получение уведомлений скрытым окном;
- Event Handle — получение уведомлений внутренним объектом приложения;
- Completion Port — порт завершения или его еще называют комплексный порт.

Далее мы рассмотрим все три механизма, начиная со средств вызова, основных качеств и заканчивая ограничениями, накладываемыми TAPI при использовании данных механизмов. Для того чтобы задать конкретный механизм уведомления в TAPI, вы должны правильно заполнить поле `dwOption` структуры `LINEINITIALIZEEXPARAMS` следующими константами:

- Hidden Window — для включения данного механизма получения сообщений и для совместимости с TAPI 1.X-приложениями используйте константу `LINEINITIALIZEEXOPTION_USEHIDDENWINDOW`;
- Event Handle — `LINEINITIALIZEEXOPTION_USEEVENT`;
- Completion Port — `LINEINITIALIZEEXOPTION_USECOMPLETIONPORT`.

Каждый из трех механизмов получения уведомлений (сообщений) работает по-разному. Так, механизм получения уведомления скрытым окном (Hidden Window) создает скрытое окно, которому будут посылаться сообщения телефонии. Механизм Event Handle создает объект получения сообщений внутри приложения и возвращает дескриптор данного объекта в поле `hEvent` структуры `LINEINITIALIZEEXPARAMS`. Наконец, механизм порта завершения инструктирует TAPI посылать сообщение в ваше приложение, используя функцию `PostQueuedCompletionStatus`. Сразу же хочется обратить внимание на то, что на разработчика ложится вся ответственность за настройку порта завершения. TAPI будет посылать сообщение порту, заданному в поле `hCompletionPort` структуры `LINEINITIALIZEEXPARAMS`. Сообщения будут помечены с ключом завершения, указанным в поле `dwCompletionKey` структуры `LINEINITIALIZEEXLine`.

При выборе механизмов уведомлений стоит учитывать и их возможные ограничения, накладываемые TAPI операционной системой. Мы упомянем только некоторые из ограничений, указанных в файле помощи TAPI. Так, например, если используете механизм получения уведомления скрытым окном, вы должны обеспечить средство считывания сообщений с очереди, причем считывание должно происходить регулярно, дабы избежать отсрочки обработки событий телефонии. Стоит заметить, что хотя основную обработку сообщений для окна Delphi делает автоматически, однако вам придется программировать отзыв на каждое рутинное TAPI-сообщение. Также при вызове `lineShutdown` TAPI автоматически уничтожит созданное окно, поэтому перед

вызовом данной функции вам придется отключать (уничтожать) средство считывания очереди.

Используя механизм Event Handle для получения сообщений, вы попадаете под ограничение, заключающееся в том, что вы не можете напрямую управлять сообщениями TAPI с помощью функций работы с процессами, нитями и волокнами Windows. Например, нельзя использовать функции `SetEvent`, `ResetEvent`, `CloseHandle` и им подобные. Данное ограничение состоит в том, что при использовании данных функций ваше приложение может повести себя самым непредсказуемым образом. Однако допускается использование функций, ожидающих поступления сообщения, вроде `WaitForSingleObject` или `MsgWaitForMultipleObjects`.

Используя механизм Completion Port для получения сообщений, вы должны вручную создавать порт завершения с помощью функции API `CreateIoCompletionPort`. После того как вы создали и настроили порт завершения на получение сообщений, ваше приложение должно считывать события телефонии, поступившие на порт завершения, используя функцию `GetQueuedCompletionStatus`. Данная функция возвратит ключ завершения, заданный в поле `dwCompletionKey` и посланный вашему приложению, а также указатель на структуру сообщения `LINEMESSAGE`, переданную в поле `lpCompletionKey`. После того как ваше приложение обработает данное сообщение, вы должны освободить занятую структурой `LINEMESSAGE` оперативную память. Поскольку ваше приложение создает порт завершения вручную (в отличие от механизма Event Handle), вы должны и закрывать данный порт самостоятельно. Порт следует закрывать после вызова функции `lineShutdown`.

Версионность TAPI

Учитывая то, что TAPI постоянно развивается, остро встает вопрос о совместимости приложений и версий TAPI. К облегчению данный вопрос решается в TAPI просто, на основе так называемого "механизма договоров". Ваше приложение и служба телефонии как бы договариваются друг с другом, какую версию TAPI использовать.

Для чего нужно договариваться о версии TAPI? Дело в том, что существуют несколько различных версий TAPI. Например, TAPI 1.3 создавалась для 16-битных систем. Версия 1.4 изначально была 32-битной и создавалась для Windows 95. Версии 2.x были уже более продвинутыми, но все еще базировались на старом TAPI, ну а в версиях 3.x и более произошла кардинальная переработка TAPI и вывод ее в технологию COM. Помимо этих версий существуют еще версии для мобильных компьютеров, например, для Windows CE (TAPI 1.5). Многообразие TAPI диктует условие, что приложение должно "договориться" со службой телефонии.

Последовательность шагов следующая: при инициализации ваше приложение договаривается о версии с TAPI в два шага.

- В первом шаге устанавливается базовая или TAPI-версия.
- Во втором шаге устанавливается расширенная версия. Базовая версия устанавливается в параметре `dwAPIVersion` функции `lineInitializeEx`.

Во многих простых приложениях TAPI вы можете не договариваться о расширенной версии, однако знайте, что расширенные функции TAPI не будут активированы службой телефонии, и вы не сможете их использовать.

Схему договора можно представить в виде следующего алгоритма. Мы предлагаем системе использовать функции своей версии TAPI, система проверяет, сможет ли она поддержать данные функции и если нет, возвращает версию TAPI, функции которой она сможет поддержать.

Мы поговорили о механизме, но не поговорили о функциях, которые и реализуют данный "механизм договора".

Итак, следующие функции представляют информацию и договариваются о версионности TAPI:

- `lineNegotiateAPIVersion` — для реализации "механизма договора" мы передаем минимальную и максимальную версии TAPI, которые приложение может поддержать в параметрах `dwAPILowVersion` и `dwAPIHighVersion`. После этого TAPI возвратит рекомендуемую версию в поле `dwAPIVersion`, причем возможны ситуации, когда эта версия будет не самой высокой, т. к. TAPI (в частности драйвер устройства) сам определяет номер TAPI-версии, функции которой он может поддержать. Однако номер версии всегда будет возвращен в диапазоне от `dwAPILowVersion` до `dwAPIHighVersion`. Если по каким-либо причинам служба телефонии не может предоставить версию, в данное поле будет возвращен 0;
- для получения информации о расширенной версии следует воспользоваться функцией `lineNegotiateExtVersion`. Данный процесс договора между приложением и TAPI подобен процессу договора функции `lineNegotiateAPIVersion`.

Определение способностей телефонии

При успешном выполнении функции `lineGetDevCaps` и `lineGetAddressCaps` возвратят вам информацию о способностях телефонии на указанном устройстве линии (коммуникационном устройстве). Данная информация будет возвращена в соответствующих структурах данных `LINEDEVCAPS` или `LINEADDRESSCAPS`. Используя информацию, содержащуюся в данных структурах, вы предпринимаете те или иные действия по отношению к заданным

устройствам линии. Стоит также отметить, что по мере выхода новых версий TAPI структуры будут пополняться все новыми и новыми данными. Так как размер структур может быть изменен в разных версиях TAPI, а информация в структуре зависит еще и от устройства, в данных структурах предусмотрено поле `dwNeededSize`, хранящее возвращаемый размер структур. Реальный размер структуры, как и в случае со структурой `VarString`, хранится в поле `dwTotalSize`. Неправильно заданный размер области памяти переменного размера может привести к ошибке `LINEERR_STRUCTURETOOSMALL` при выполнении данных функций. Порядок выделения памяти точно такой же, как и в случае со структурой `VarString`.

Следует заметить, что, используя `lineGetDevCaps` для получения информации о линии, вы получите информацию об адресах линии. Устройство линии может включать множество адресов. Количество адресов будет возвращено в одном из полей структуры `LINEDEVCAPS`. Способности адреса могут измениться так же, как способности линии. Для считывания способностей адреса устройства линии (коммуникационного устройства) вы должны использовать функцию `lineGetAddressCaps` для каждой доступной комбинации `dwDeviceID/dwAddressID`.

Стоит отметить, что дополнительные данные, возвращаемые функциями, записываются в область памяти после области, занимаемой самой структурой `LINEDEVCAPS` или `LINEADDRESSCAPS`. То есть в памяти сначала идет блок, равный `SizeOf(LINEDEVCAPS` или `LINEADDRESSCAPS)` плюс дополнительные данные. В дополнительные данные входит различная информация, не представленная в самой структуре. Например, в данном блоке передается наименование устройства линии. Предвидя резонный вопрос: как считать данную информацию, ответим, что для этого в структурах предусмотрены специальные поля, заканчивающиеся на `Size` и `Offset`. Поясним их назначение, в полях, названия которых заканчивается на Size, хранится информация о размере специфичного блока данных; а в полях, заканчивающихся на Offset, — смещение в байтах от начала структуры. Например, чтобы получить название устройства линии, нужно считать массив данных размером `LineDevCaps.dwLineNameSize`, начиная с блока памяти, равного `PChar(LineDevCaps)` плюс `LineDevCaps.dwLineNameOffset`.

Обратите также внимание на то, что старые приложения (использующие старые версии TAPI, особенно 1.x) не включают подобные поля в состав структур `LINEDEVCAPS` и `LINEADDRESSCAPS`.

Примечание

Так как структуры данных различаются от версии к версии, на программиста ложится задача правильного заполнения полей данных структур.