

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

Разработка ПО для портативных компьютеров (КПК)

Введение

Жерздев С.В.

КПК

- Самым первым КПК был Newton MessagePad от компании Apple. Впервые мир узнал о нем в 1992 году. Тогдашний президент Apple Джон Скалли впервые употребил понятие PDA (Personal Digital Assistant - персональный цифровой помощник). У нас аббревиатуры PDA или ПЦП не прижились, и для названия этого класса устройств чаще всего используется термин "карманный персональный компьютер", или КПК.

КПК

- Карманный персональный компьютер заметно отличается как от простейших органайзеров с джентльменским набором функций типа планировщика, записной книжки, будильника и калькулятора, так и от компактных представителей персональных компьютеров - ноутбуков. КПК превосходят органайзеры прежде всего функциональностью. По своим возможностям современные КПК уже серьезно догоняют ноутбуки. И все же это совсем разные устройства, прежде всего по типу операционной системы, ну и по размерам, конечно. А еще КПК намного дольше работают от батарей.

Особенности программирования портативных устройств

Особенности программирования портативных устройств

- В настоящее время все большую популярность приобретают различного вида мобильные и портативные устройства, включая сотовые телефоны и коммуникаторы, карманные персональные компьютеры (КПК) и системы навигации. Хотя все они содержат в себе в том или ином виде универсальное вычислительное устройство, архитектура их может существенно отличаться от архитектуры персональных компьютеров (ПК).

Размер экрана

- Из соображений эргономики физические размеры экрана ограничены диагональю 3,5-4 дюйма, а типичное разрешение составляет 160*160, 320*240 или 320*320 пиксель.
- Необходимо обеспечивать баланс между информационной насыщенностью и уровнем заполнения экрана, но при этом в большинстве случаев разрешение экрана может зависеть от конкретной модели и не известно заранее.

Быстрый отклик

- На мобильных устройствах, таких как КПК, приложение может использоваться 15-20 раз по несколько секунд в течение дня. Скорость приложений имеет приоритет при разработке.
- Существенное влияние на общую эффективность оказывает не только скорость выполнения кода, но и удобство взаимодействия пользователя с интерфейсом приложения.
- Для увеличения производительности следует минимизировать количество перемещений между окнами, обрабатываемых диалогов и т.п.

Взаимодействие с ПК

- Многие мобильные устройства обеспечивают средства взаимодействия с внешним миром (через кабель, инфракрасные или беспроводные интерфейсы) и для многих приложений важной является задача обеспечения взаимодействия, передачи и синхронизации данных с соответствующими приложениями на ПК.
- В некоторых случаях это требует разработки программных средств не только для мобильного устройства, но и для ПК, например, для упрощения, сжатия или предварительной ресурсоемкой обработки данных перед передачей.

Ввод данных

- Портативные устройства в силу своих габаритов не могут обеспечить пользователя полноразмерными устройствами ввода — клавиатурой и мышью. Как правило, устройство оснащается упрощенной или виртуальной клавиатурой и/или сенсорным экраном.
- Учитывая, что эти средства не обеспечивают достаточного удобства, следует минимизировать объем вводимой пользователем информации.

Питание

- Портативные устройства обеспечиваются, как правило, источником энергии существенно ограниченной емкости. Соответственно, ресурсоемкие задачи, требующие большого объема вычислений, следует по возможности выносить для решения сопутствующим ПО на стационарных ПК.

Память

- Портативные устройства являются ограниченными по объему доступной памяти, как для времени исполнения, так и для хранения данных. Типичное значение доступной памяти для КПК — от 512 Кб до 128 Мб.
- Существенной является оптимизация применяемых алгоритмов и программ по следующим приоритетным направлениям:
 - объем памяти, используемый при работе;
 - скорость;
 - объем кода.

Файловая система

- По причине ограниченного объема памяти для хранения данных и для более эффективной синхронизации с ПК, портативные устройства редко используют традиционные файловые системы.
- Типичные свойства, обеспечиваемые ОС — доступ и установка атрибутов отдельных записей, а не всех файлов для обеспечения частичная синхронизация и работа с записями по месту, без предварительной загрузки и последующей записи.

Сетевые средства

- Как правило, сетевое соединение или недоступно или является “дорогостоящим”. Причины:
 - непостоянное соединение;
 - возможность соединения ограничена географически;
 - ограниченная полоса пропускания;
 - применение энергоемких беспроводных технологий;
 - высокая стоимость трафика;
 - все вышеперечисленное в любых комбинациях.
- Следствие — требуется по возможности минимизировать необходимость присутствия в сети, количество соединений, объем сетевого трафика.

Наиболее популярные платформы КПК



PalmOS



PalmOS

- Первоначальная разработка компании US Robotics, затем компания продана 3COM, а потом и вообще подразделилась на два подразделения - одно выпускает сами КПК, другое подразделение - разрабатывает операционную систему и софт.
- Отличительная особенность операционной системы - небольшое использование ресурсов (памяти и процессора). Более низкая цена по сравнению с PocketPC для КПК с одинаковым функционалом. Ранее, до появления OS5, PalmOS проигрывала по скорости вычислений, мультимедийным возможностям и графике.

PalmOS

- Использование в качестве органайзера - удобное, время работы от аккумуляторов - от 10 до 25 часов в зависимости от модели. Распознавание ввода - Граффити, но у некоторых КПК есть и клавиатура. Ввод букв осуществляется росчерками, напоминающими написание букв алфавита с небольшими упрощениями в определенной области экрана.

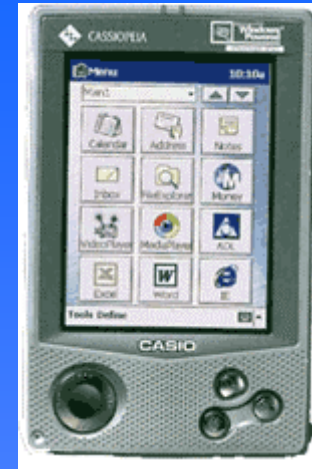
PalmOS

- КПК на базе PalmOS разрабатываются следующими компаниями:
 - Palm (законодатель КПК на базе PalmOS)
 - Sony (модели CLIE)
 - Handspring (компания куплена Palm-ом)
 - Fossil (наручный компьютер в виде часов)
 - Kyocera (смартфоны)
 - Samsung (смартфоны)
 - Acer (также выпускает и PocketPC)
 - Garmin (КПК со встроенным GPS приемником)
 - Symbol

PalmOS

- Наиболее развитые аппараты выпускает Sony, встраивая цифровые камеры, клавиатуры, используя поворотные дисплеи высокого разрешения. Отличительная особенность - Sony использует карты MemoryStick в качестве внешней памяти, они не очень быстрые, цена не дешевая, удобство представляет только для тех, кто уже имеет у себя что-нибудь использующее эти карты. Palm постоянно выпускает конкурентоспособные устройства, последнее время большой упор делается на коммуникационные возможности КПК.

WindowsCE, PocketPC, CE.NET



WindowsCE, PocketPC, CE.NET

- По сути, все эти три названия объединяют один класс устройств с операционной системой WindowsCE. PocketPC и CE.NET это просто названия последних версий WindowsCE. WindowsCE 2.0 и более ранние разработки полностью устарели, используется только в старых КПК и интереса уже не представляет. PocketPC - наиболее популярная ОС на данный момент в данной категории.

WindowsCE, PocketPC, CE.NET

- Операционная система WindowsCE задумывалась Microsoft как облегченная версия Windows 95, рассчитанная на использование в портативных компьютерах. Изначально ориентированная на самые различные аппаратные платформы и процессоры, WindowsCE позволяет легко менять конфигурации портативных устройств и применяется не только в PDA, но и в субноутбуках, автомобильных компьютерах, Web-планшетах и т. д.

WindowsCE, PocketPC, CE.NET

- В отличие от PalmOS, WindowsCE - это многофункциональная система, имеющая модульную основу и широкие возможности расширения. Операционная система Microsoft является многозадачной, может работать с мультимедийными данными и цветными экранами (до 65 тысяч цветов). При всех несомненных достоинствах минусом этой ОС являются завышенные требования к оперативной памяти, "неторопливость" и неприспособленность стандартного интерфейса Windows под нужды мобильных пользователей.

WindowsCE, PocketPC, CE.NET

- Производят КПК на базе PocketPC:
 - HP
 - Compaq
 - DELL
 - Rover
 - Toshiba
 - Asus
 - Acer
 - Casio.

WindowsCE, PocketPC, CE.NET

- Программы для этого класса КПК более мощные, из-за большей памяти и больших скоростей процессора возможно создание достаточно мощных и серьезных приложений. Цена на них, более высокая, по сравнению с Palm-ами. Менее удобны они в качестве органайзеров. Есть еще один недостаток - время автономной работы, оно тоже меньше. Работа с устройством напоминает на обычную работу в Windows, поэтому пользователю достаточно легко ориентироваться.

EPOC, Symbian



ЕРОС, Symbian

- Операционная система ЕРОС, используемая во всех моделях карманных компьютеров, выпущенных английской компанией Psion PLC, разрабатывается фирмой Symbian - совместным предприятием, организованным Psion в сотрудничестве с ведущими производителями сотовых телефонов, и позиционируется в качестве индустриального стандарта для мобильных аппаратов следующего поколения. Текущая версия этой ОС - ЕРОС32 ER5. Операционная система ЕРОС - многозадачная, оконная, поддерживает работу с цветными экранами.

ЕРОС, Symbian

- Компьютеры на базе операционной системы ЕРОС под торговой маркой Psion выпускала только английская компания Psion PLC. Эти PDA разительно отличаются от Palm. Во-первых, все они, кроме Psion Revo, имеют достаточно большие как для карманных устройств размеры и массу, но в отличие от palm-size PDA снабжены полнофункциональной клавиатурой с 53-58 клавишами, на которой после определенной тренировки можно печатать даже вслепую.

ЕРОС, Symbian

- Во-вторых, операционная система ЕРОС больше похожа на ОС, используемые в настольных ПК (оконная среда, иерархическая папочная структура, развитая система меню и short-cut), то есть переучивание при переходе на использование карманного компьютера в данном случае минимальное. Да и приложения, очень похожие на своих старших настольных братьев, способствуют такому легкому переходу.

ЕРОС, Symbian

- Благодаря изначальной ориентации на использование в сотовой связи, в этой операционной системе очень хорошо реализованы функции взаимодействия с мобильными телефонами последних моделей производства Nokia, Siemens, Ericsson и Motorola.

ЕРОС, Symbian

- Сегодня это полнофункциональная операционная система, созданная с учетом всех требований телекоммуникационной индустрии и большинства современных стандартов и протоколов, таких как Bluetooth, GPRS и т.п. Ядро системы - многозадачное, высокопроизводительное и исключительно компактное - может быть без больших затрат перенесено практически на любую платформу. Полная поддержка Unicode позволяет без проблем адаптировать систему для любого языка, гибкие механизмы расширения позволяют решить все проблемы с кодировками почты, Web и т.п.

ЕРОС, Symbian

- Начиная с версии 6.0, введена диверсификация на "семейства" устройств (reference design). На данный момент определены 3 класса устройств: безклавиатурные КПК с форм-фактором, напоминающим Palm и Rocket PC, клавиатурные коммуникаторы и смартфоны. Все три семейства используют одно ядро, различия в основном сводятся к пользовательскому интерфейсу, форм-фактору, отсутствию/наличию сенсорного экрана и т.п.

ЕРОС, Symbian

- Symbian OS поддерживает большинство стандартов, принятых в индустрии мобильной связи: GSM/EGSM, GPRS, HSCSD, CDMA.
- Больше всего эта ОС используется в коммуникаторах/смартфонах.

Другие



Другие

- Мир КПК растет и некоторые производители не хотят покупать лицензии на установку системы, они пытаются сделать свою ОС, которая снизит общую стоимость КПК. Однако, хорошую ОС создать очень сложно. Более того, пока под нее не будет достаточного количества программ, она не будет представлять интереса. Пример - RocketViewer, PenbexOS, Mine OS (EZnow) и другие малоизвестные OS для КПК.

Другие

- В последнее время с ростом популярности ОС Linux все больше появляется и КПК на базе такой ОС (Sharp, Zaurus). Тем не менее, при всех преимуществах открытого ПО они пока захватили очень небольшую долю рынка.

Литература

- 1. PocketPC 2003 SDK. - MSDN.
- 2. Боулинг Д. Windows CE: лилипут на арене - PC Magazine, 10.1997, p. 285
- 3. Боулинг Д. Программирование для Windows CE - PC Magazine/RE, 10.1999.
- 4. Palm OS Programmer's Companion - Palm OS SDK.
- 5. Palm OS Development Documentation - Palm OS SDK.
- 6. Таскер М. Основы Си++ API EPOC - Symbian, 1999

Программное обеспечение

- 1. Microsoft Embedded Visual C++ v.4.
- 2. Microsoft Pocket PC 2003 SDK
- 3. GNU C compiler for Palm OS.
- 4. GNU Source-Level Debugger.
- 5. GNU linker.
- 6. PilRC.
- 7. Palm OS Emulator.
- 8. Palm SDK.

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

Разработка ПО для портативных компьютеров (КПК)

Palm OS

Жерздев С.В.

PalmOS

- Большую роль в распространении КПК сыграли устройства на базе операционной системы PalmOS, которая появилась в 1996 году. За последующие 6 лет в свет вышло более 5 версий PalmOS (1.0 - 4.1). Большим изменениям PalmOS никогда не подвергалась, причины это кроются в том, что для PalmOS создавалось очень много программ (это продолжается и по сей день). Вторая причина - аппаратная часть КПК на базе PalmOS долгое время оставалась неизменной.

PalmOS

- Пятая версия PalmOS, которая появилась относительно недавно, полностью переработана для работы на новой аппаратуре, поддержки средств мультимедиа. КПК на базе PalmOS 5 еще не успели распространиться.

Основные характеристики устройств

- Процессор Motorola DragonBall
- Объем памяти (RAM) от 1 до 8 Мб
- Экран 160*160 (в некоторых моделях 320*320), сенсорный, 2 / 4 / 16 градаций серого или цветной
- Несколько функциональных клавиш
- Ввод текста с помощью системы Граффити или виртуальной клавиатуры
- Дополнительная аппаратура - инфракрасный порт, динамик, СОМ.

PC connectivity

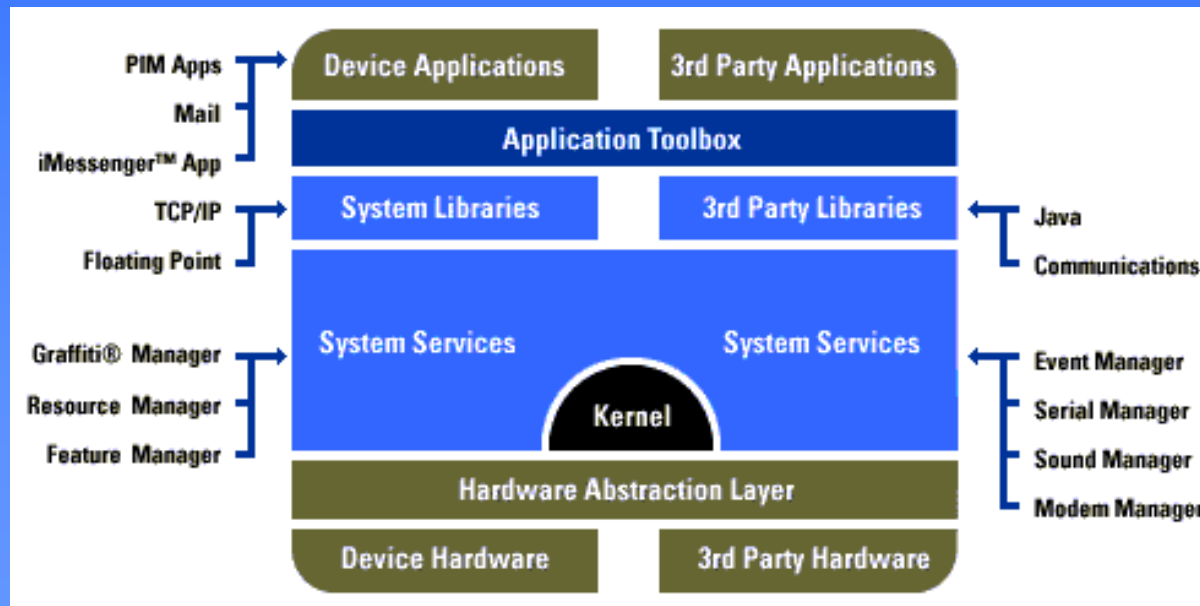
- Важную часть платформы составляет система взаимодействия с ПК - PC connectivity. Устройства поставляются с подставкой (cradle), которая подключается к ПК и ПО, которое обеспечивает сохранение и синхронизацию данных "одной кнопкой".
- Многие приложения PalmOS имеют соответствующие приложения для ПК. Для синхронизации данных применяются дополнения (conduit) к технологии HotSync, которые активизируются по нажатию кнопки синхронизации.

Особенности программирования

- Приложения Palm OS, как правило, однопоточные событийно-управляемые. Хотя ОС основана на ядре с поддержкой многозадачности (микро ядро AMX, разработано фирмой Kodak), одновременно может быть запущено одно приложение. Когда пользователь запускает другое приложение, то предыдущее текущее приложение завершает свою работу (все данные автоматически сохраняются).

Особенности программирования

- PalmOS состоит из модулей (Manager) и библиотек (Library), например Memory Manager - модуль управления памятью, Data Manager - модуль управления файловой системой, Библиотека TCP/IP.



Особенности программирования

- Каждое приложение имеет функцию PilotMain, эквивалент main() в программах на С. Для запуска приложения система вызывает PilotMain и передает код запуска (launch code). Код запуска может определять, что приложение должно стать активным и отобразить интерфейс пользователя (обычный запуск), или что приложение должно выполнить небольшую задачу и завершить работу без отображения интерфейса. Основная задача функции PilotMain - принять код запуска и ответить на него.

Особенности программирования

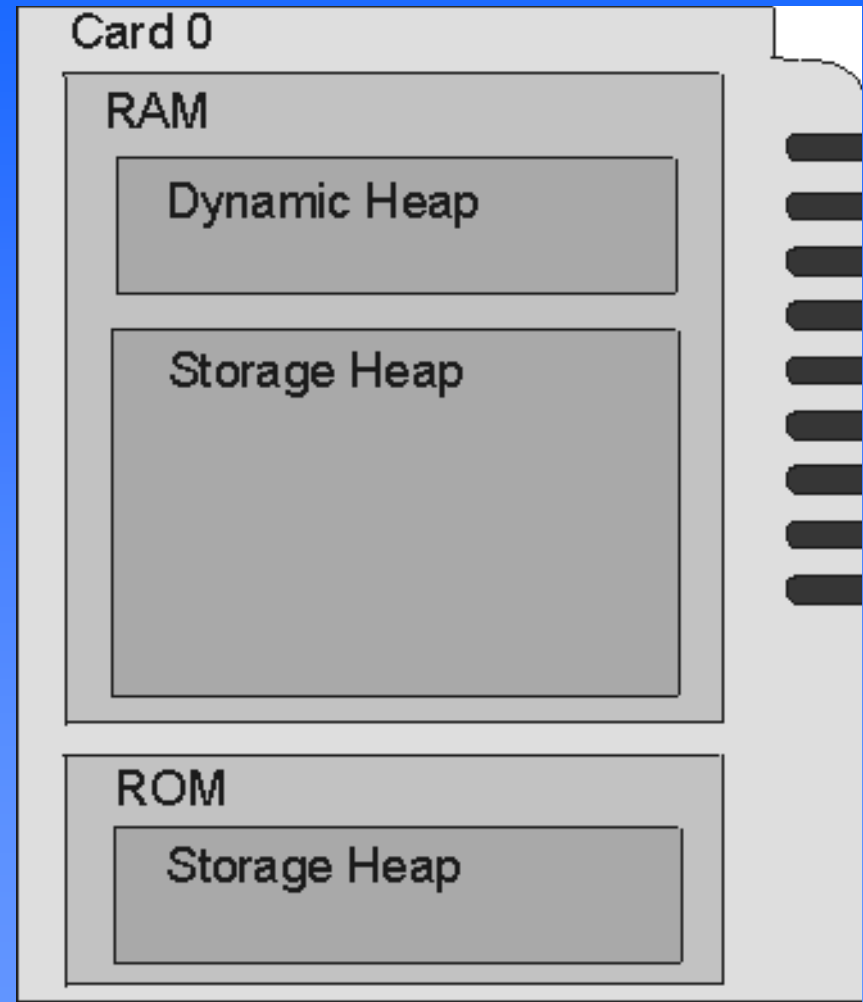
- Приложение может получать различные уведомления о системных событиях, например: вставка/извлечение модуля расширения, удаление файла, изменение настроек экрана или сети, изменение времени, подключение/отключение от сети, переход в режим "сна", легкий рестарт, синхронизация. Уведомления "приходят" в приложения в виде кода запуска (launch code).

Особенности программирования

- Поскольку Palm OS является событийно-ориентированной ОС, приложения Palm OS содержат цикл обработки событий (event loop). Однако, он используется только при обычном запуске приложения.

Память и файловая система

- Физически память расположена на картах (Card) памяти, которые нумеруются - 0, 1, Каждая карта памяти может иметь RAM и ROM сегменты (RAM - random access memory, ROM - read only memory).



Память и файловая система

- PalmOS делит все пространство RAM сегмента на Dynamic Heap и Storage Heap. Storage Heap это эквивалент диска (HDD), там находятся только файлы. В Dynamic Heap находятся все динамические объекты приложения, операционной системы, библиотек, модулей, а также стек. В PalmOS код приложения не загружается в Dynamic Heap для выполнения, выполнение происходит по месту (inplace), т.е. все программы, как и сама PalmOS, всегда выполняются прямо из Storage Heap. Это прозрачно для приложений.

Память и файловая система

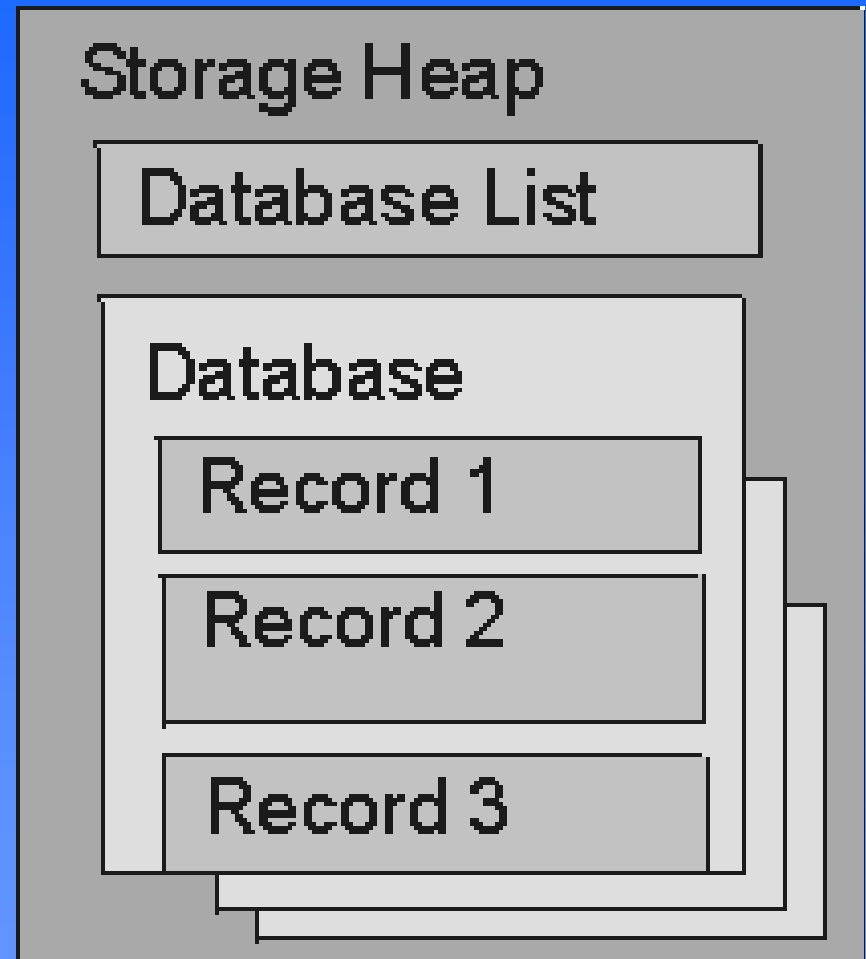
- Т.к. ROM это сегмент памяти "только для чтения", то соответственно там находится только файлы. Например, большинство моделей КПК имеет карту памяти - Card0, и в ROM сегменте этой карты находятся файлы операционной системы со встроенными приложениями. Приложение может получить список всех карт, и указать с какой картой оно будет работать.

Память и файловая система

- Размеры Dynamic Heap и Storage Heap определяются операционной системой при инициализации и зависят от общего размера памяти в КПК. Обычно размер Dynamic Heap 32-256Кб.

Память и файловая система

- В PalmOS есть своя файловая система. Пользователь в PalmOS не работает ни с папками, ни с файлами. Пользователь "видит" только приложения и документы с которыми работает данное приложение. Главная особенность файловой системы это - отсутствие директорий.



Память и файловая система

- В PalmOS файл называется database. Как и в любой ОС у него есть имя, тип, размер, другие атрибуты. Database в свою очередь разбит на records (Запись). У Записи есть такие атрибуты как: порядковый номер, размер, и др. В Записях уже непосредственно хранятся данные.
- Можно сказать, что database это аналог директории, а record это аналог файла т.к. приложение сначала открывает database а потом в ней открывает record на чтение или запись.

Интерфейс пользователя

- ОС поддерживает графический интерфейс пользователя для приложений (диалоги, кнопки, списки, шрифты, ...). Многие приложения Palm OS содержат интерфейс пользователя, состоящий из форм (аналог окон в ОС ПК).

The screenshot shows a window titled "Mailer - Edit text" with a "General" tab. The interface includes a "To:" field with the text "text" and a dotted line for additional input. Below this are "Set Time:" and "Set Date:" fields, each with a dotted box containing "6:36 pm" and "1/22/02" respectively. A "Private:" field has a checked checkbox. The "Priority:" field consists of five numbered buttons (1-5), with button "1" highlighted. A "Send" button is located at the bottom right.

Интерфейс пользователя

- Базовые элементы GUI - это Форма (Form), Диалог, Меню. Плюс традиционные элементы любого графического интерфейса: кнопки, флажки (checkbox), списки, поля с текстом, селекторы, таблицы, картинки. Интерфейс создается в виде словесного описания в файле ресурсов (см. пример приложения). В этом файле описывается, какие формы будет показывать приложение и какие элементы (кнопки, надписи ...) будут на формах.

Системные средства

- Инфракрасный порт, последовательный порт, и сеть - это 3 основных типа коммуникаций в PalmOS. Приложение может пользоваться инфракрасным портом на высоком уровне - это пересылка файлов между КПК, и на низком уровне эта работа с портом напрямую.
- С помощью TCP/IP протокола можно работать с интернетом или локальной сетью. В PalmOS реализованы Berkley Sockets функции, можно работать в асинхронном или синхронном режимах (не блокируемые вызовы и блокируемые).

Системные средства

- Операционная система PalmOS автоматически управляет питанием компьютера. Например, по истечении одной минуты бездействия, КПК выключается; если приложение не выполняет никакой работы, то процессор приостанавливает свою работу. Для приложений это прозрачно.

Системные средства

- Приложение может самостоятельно сделать легкую перегрузку операционной системе.
- Легкий рестарт - это отчистка Dynamic Heap памяти, проверка целостности Storage Heap, и дефрагментация всей памяти.
- Полный Рестарт это обнуление всей памяти (уничтожение всей файловой системы) и создание файловой системы заново, потом туда копируются приложения операционной системы и производится установка настроек, в которой принимает участие пользователь.

Создание приложений,
Инструментарий, POSE

Создание приложений

- Программы для PalmOS обычно создаются на С (С++), однако существуют компиляторы для других языков: Java, Pascal, Basic, Assembler. (для запуска Java приложений нужно будет дополнительно установить Java Машину).
- Кроме того, существуют средства разработки on-board, работающие непосредственно на устройстве PalmOS и позволяющие разрабатывать небольшие приложения без использования ПК. Наиболее популярным из таких средств является OnBoardC.

Инструментарий

- Самая популярная коммерческая среда разработки это CodeWarrior, также есть Falch.Net, VFDIDE, PilotMAG - все эти "визуальные" среды разработки имеют редактор кода с синтаксической подсветкой кода и другие функции. (управление проектом, встроенный отладчик, редактор ресурсов, ...).
- Самая популярная некоммерческая среда это PRC-Tools (компилятор gcc под процессор Motorola DragonBall 68K). Этот инструментарий бесплатный, популярный, и официально поддерживается компанией PalmSource, Inc.

Инструментарий

- Чтобы создать простое приложение для PalmOS с помощью PRC-Tools, необходимо установить CygWin(эмуляция unix среды в Windows), PRC-Tools (gcc компилятор), PilRc for Win32 (компилятор ресурсов), PalmOS SDK 3.5, PalmOS Emulator.
- Для редактирования файла ресурсов удобно использовать визуальный редактор pibuilder.

POSE

- Компания Palm Computing (ныне PalmSource, Inc) создала эмулятор операционной системы PalmOS (PalmOS Emulator - POSE), для тестирования приложений на настольном компьютере. После запуска эмулятора на экране появляется "картинка" КПК - можно мышкой кликать на экране КПК как стилусом (даже писать символы граффити), можно запускать приложения.

Преимущества POSE

- Процесс загрузки приложения в POSE намного быстрее, чем в реальный КПК, что намного ускоряет процесс тестирования. Не нужно каждый раз синхронизироваться.
- В POSE можно сохранять текущее состояние КПК, а потом к нему возвращаться.
- В POSE можно делать скриншоты.
- С помощью POSE можно увидеть различные КПК с PalmOS разных версий с различным размером памяти, без наличия самого КПК.
- В POSE можно регулировать уровень системных ошибок, которые следует обрабатывать

Простое приложение

- Простое приложение состоит из 3 файлов: исходный код на С или другом языке, заголовочный файл и файл ресурсов приложения.
- Каждое Palm приложение имеет уникальный CreatorID, это 4 байта, например "MyAp". Палм использует CrID для того чтобы различать приложения и его файлы.
- После компиляции и сборки получается файл с расширением "prc" - это файл приложения для PalmOS, его можно поместить в эмулятор для тестирования (или установить в КПК через HotSync).

Технологическая цепочка

- Для создания простейшего приложения с использованием prc-tools необходимо выполнить следующие шаги.

Создание исходных файлов.

- В простейшем случае файл main.c имеет вид

```
#include <PalmOS.h>
```

```
#include "main.h"
```

```
UInt32 PilotMain (UInt16 cmd, void *cmdPBP,  
    UInt16 launchFlags)
```

```
{  
    // Launch code sent by the launcher.  
    if (cmd == sysAppLaunchCmdNormalLaunch)  
    {  
        FrmAlert (alertId);  
    }  
}
```

Создание исходных файлов.

- Заголовочный файл `main.h` содержит определения идентификаторов ресурсов.

```
#define alertId 1000
```

Создание файла ресурсов hello.rcp

```
#include "main.h"

ALERT ID alertId
    INFORMATION
    DEFAULTBUTTON 1
BEGIN
    TITLE "My Alert"
    MESSAGE "Hello, World!"
    BUTTONS "Hi!" "Get off..."
END
```

Компиляция и сборка

- Создание объектного файла

```
m68k-palmos-gcc -c -O2 -Wall main.c
```

- Создание исполняемого файла

```
m68k-palmos-gcc main.o -O2 -Wall -o hello
```

- Создание двоичного представления ресурсов

```
plrc hello.rcp
```

- Сборка prc-файла для загрузки на устройство

```
build-prc hello.prc hello SerZ hello *.bin
```

Автоматизация

- Все этапы компиляции и сборки можно автоматизировать с помощью makefile

```
# compiler commands
CC          = m68k-palmos-gcc
PILRC       = pilrc
BUILDPRC    = build-prc
# source / outputs
OBJS        = main.o
EXEC        = hello
CRID        = SerZ
# compiler flags
CCFLAGS     = -O2 -Wall
```


Автоматизация

```
# compile requirements
$(EXEC).prc: $(EXEC) bin.stamp
    $(BUILDPRC) $(EXEC).prc $(EXEC) $(CRID)
    $(EXEC) *.bin
make clean

$(EXEC) : $(OBJS)
    $(CC) $(OBJS) $(CCFLAGS) -o $(EXEC)

bin.stamp: $(EXEC).rcp
    $(PILRC) -q $(EXEC).rcp
```

Автоматизация

```
# compile rules
.SUFFIXES: .c .o

.c.o:
    $(CC) -c $(CCFLAGS) $<

# clean-up funtions
clean:
    rm -f *. [oa] *.bin *.grc *~ $(EXEC)

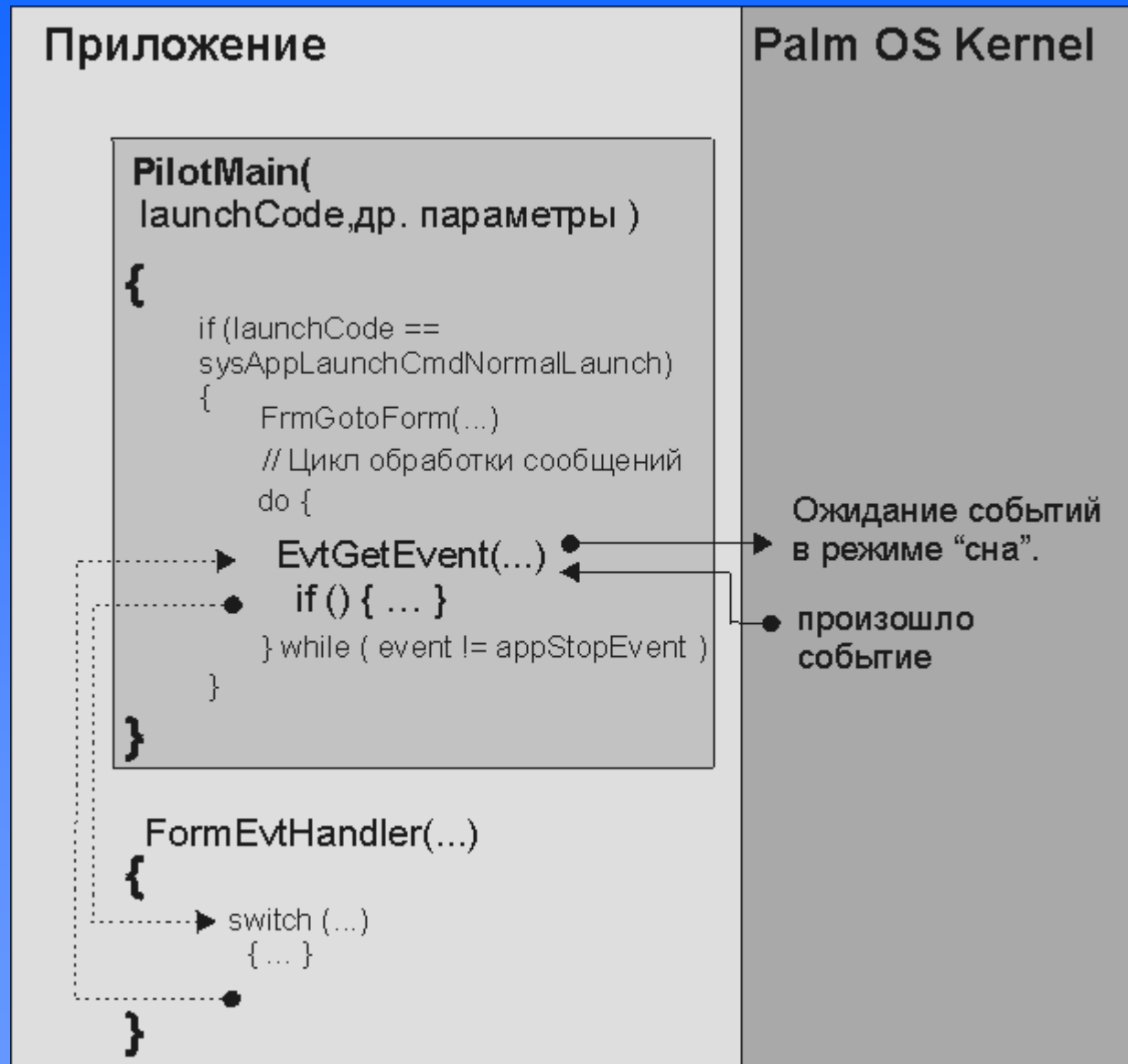
distclean:
    rm -f *. [oa] *.bin *. [pg]rc *~ $(EXEC)
```

Пример приложения PalmOS

Пример приложения PalmOS

- Точка входа в приложение - функция PilotMain.
- UInt32 PilotMain (UInt16 launchCode, void *cmdPBP, UInt16 launchFlags)
- На вход этой функции передается несколько параметров, самый важный параметр это - LaunchCode - код запуска приложения. SysAppLaunchCmdNormalLaunch это код нормального запуска - приложение должно показать свое главное окно.

Общая схема работы приложения



Интерфейс

- Интерфейс создается в виде словесного описания в файле ресурсов. В этом файле описывается, какие формы будет показывать приложение и какие элементы (кнопки, надписи ...) будут на формах. Обычно приложение содержит несколько Форм и одно Меню, форма может занимать только часть экрана. Все формы модальные, т.е. пока одна форма активна, другая форма не может обрабатывать события.

События

- События которые происходят на форме (нажатие кнопки, ввод буквы, команда меню выделение строки списка) приходят в процедуру - обработчик сообщений формы. В это процедуре можно определить тип события, и считать данные с элементов на форме (например тест из текстового поля или состояние флажка)

Пример приложения

- Рассмотрим пример простого приложения - `example1.prc`. Оно состоит из одной формы. На Форме расположена кнопка, надпись и текстовое поле, к форме прикреплено меню. При нажатии на кнопку появляется сообщение. Проект приложения состоит из 4х файлов:
 - `Example1.c` - исходный код на языке C.
 - `Example1.h` - имена и номера элементов формы
 - `Example1.rcp` - описание ресурсов приложения (форма, меню, кнопка, диалог сообщения).
 - `Makefile` - для сборки приложения.

Файл example1.c

```
#include <PalmOS.h>
#include "hw.h"
static FormPtr gpForm;
```

Файл example1.c

```
Boolean Form1_HandleEvent(EventPtr event)
{
    switch(event->eType)    {
    case ctlSelectEvent:
        if ( event->data.ctlEnter.controlID == Button1)
            FrmCustomAlert(Alert1, "MessageBox !", "", "");
        break;
    case menuEvent:
        if ( event->data.menu.itemID == IDM_about)
            FrmCustomAlert(Alert1, "This is Example", "", "");
        break;
    case frmOpenEvent:
        gpForm=FrmGetActiveForm();
        FrmDrawForm(gpForm);
        break;
```

Файл example1.c

```
case frmCloseEvent:
    FrmEraseForm(gpForm);
    FrmDeleteForm(gpForm);
    break;

default:
    return false;

}
return true;
}
```

Файл example1.c

```
static Boolean ApplicationHandleEvent(EventPtr
    event)
{
    FormPtr frm;
    Boolean handled=false;
    switch(event->eType)    {
    case frmLoadEvent:
        frm = FrmInitForm (event->data.frmLoad.formID);
        FrmSetActiveForm (frm);
        FrmSetEventHandler(frm , Form1_HandleEvent);
        handled = true;
        break;
    }
    return handled;
}
```

Файл example1.c

```
UInt32 PilotMain(UInt16 launchCode, void *cmdPBP, UInt16
    launchFlags)
{
    EventType event;
    UInt16 error;

    if (launchCode==sysAppLaunchCmdNormalLaunch) {
        FrmGotoForm(Form1);
        do {
            EvtGetEvent(&event, evtWaitForever);
            if (!SysHandleEvent(&event))
                if (!MenuHandleEvent(0, &event, &error))
                    if (!ApplicationHandleEvent(&event))
                        FrmDispatchEvent(&event);
        }
        while (event.eType!=appStopEvent);
        FrmCloseAllForms();
    }
    return 1; }
```

Файл example1.h

```
#define Form1      1000
#define Button1    1003
#define Edit1      1006
#define MenuBar1   1000
#define Alert1     1102

#define IDM_about   1001
#define IDB_Edit1   1002
```

Файл example1.rcp

```
#include "hw.h"

FORM ID Form1 AT (0 0 160 160) FRAME USABLE
MENUID MenuBar1
    BEGIN
        TITLE "Example 1"
        LABEL "This is an example" AUTOID AT (10 10+5) FONT 1
        LABEL "of small GUI programm" AUTOID AT (PREVLEFT
PREVBOTTOM+5) FONT 2
        FIELD ID IDB_Edit1 AT (PREVLEFT PREVBOTTOM+5 110 20)
MAXCHARS 50 UNDERLINED FONT 2
        BUTTON "Hello" ID Button1 AT (10 80 AUTO AUTO)
    END
```

Файл example1.rcp

```
MENU ID MenuBar1
    BEGIN
        PULLDOWN "Help"
        BEGIN
            MENUITEM "About" ID IDM_about "A"
        END
    END

END

ALERT ID Alert1
    INFORMATION
    BEGIN
        TITLE "Example 1"
        MESSAGE "^1 ^2 ^3"
        BUTTONS "Ok"
    END
```


Файл makefile

```
CC=m68k-palmos-gcc.exe
PILRC=pilrc.exe
BUILDPRC=build-prc
OBJRES=m68k-palmos-obj-res
GRC=*.grc
CFLAGS = -O2
all: example1
example1: example1.bin resources
    ls *.grc|$(BUILDPRC) example1.prc Test -v 1.0 *.bin *.grc
example1.bin: example1.o
    $(CC) -o $@ $<
    $(OBJRES) $@
example1.o: example1.c
    $(CC) $(CFLAGS) -c $<
resources: example1.rcp
    $(PILRC) $<
clean:
    rm *.bin *.o *.grc
```

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

Разработка ПО для портативных компьютеров (КПК)

Palm OS (2)

Жерздев С.В.

Коды запуска

- Приложение запускается с получением кода запуска (launch code). Коды запуска обеспечивают взаимодействие между Palm OS и приложением (или между приложениями).
- Обычно, приложение запускается по команде пользователя в экране запуска приложений. В этом случае система формирует код запуска `sysAppLaunchCmdNormalLaunch`, который требует от приложения осуществить полный запуск с отображением интерфейса пользователя.

Коды запуска

- Другие коды запуска требует от приложения выполнения некоторых действий, необязательно сопровождающихся отображением на экране. Например, для обработки глобального поиска приложение осуществляет только поиск в своих базах данных.

Коды запуска

- Каждый код запуска дополняется двумя видами информации:
 - Блоком параметров, указателем на структуру, содержащую информацию для обработки соответствующего кода запуска.
 - Флаги запуска (launch flags) уточняют требуемое поведение приложения. Например, флаг может определять, следует ли приложению отобразить интерфейс пользователя.
- Приложение должно обработать коды запуска в функции PilotMain, которая является точкой входа в приложение.

Коды запуска

- Получив код запуска, приложение должно проверить, обрабатывает ли оно данный конкретный код. Например, только приложения с текстовыми данными должны отвечать на код запуска с требованием поиска строки. Если приложение не обрабатывает данный код, оно должно завершиться без каких-либо дополнительных действий. Иначе, приложение должно немедленно выполнить требуемое действие и завершить работу.

Нормальный запуск

- Получив код запуска `sysAppLaunchCmdNormalLaunch`, приложение выполняет начальную инициализацию, переходит в цикл обработки событий и затем завершает работу.

Нормальный запуск

- В процессе инициализации приложению следует выполнить три действия:
 - Получить системные настройки (например, формат представления чисел, даты и времени) и занести их значения в глобальные переменные, используемые во всем приложении.
 - Найти базу данных приложения по идентификатору создателя (creator type). Если она не существует, создать и инициализировать ее.
 - Получить настройки приложения и инициализировать глобальные переменные.

Пример инициализации

```
static UInt16 StartApplication (void)
{
    UInt16 error = 0;
    Err err = 0;
    UInt16 mode;
    DateTimeType dateTime;
    DatebookPreferenceType prefs;
    SystemPreferencesType sysPrefs;
    UInt16 prefsSize;
```

Пример инициализации

```
// Step 1: Get system-wide preferences.
PrefGetPreferences (&sysPrefs);
// Determine if secret records should be displayed.
HideSecretRecords = sysPrefs.hideSecretRecords;

if (HideSecretRecords)
    mode = dmModeReadWrite;
else
    mode = dmModeReadWrite | dmModeShowSecret;

// Get the time formats from the system preferences.
TimeFormat = sysPrefs.timeFormat;

// Get the date formats from the system preferences.
LongDateFormat = sysPrefs.longDateFormat;
ShortDateFormat = sysPrefs.dateFormat;
...
```

Пример инициализации

```
// Step 2. Find the application's data file. If it doesn't
// exist, create it.
ApptDB = DmOpenDatabaseByTypeCreator(datebookDBType,
sysFileCDatebook, mode);
if (! ApptDB)
{
    error = DmCreateDatabase (0, datebookDBName,
sysFileCDatebook,
datebookDBType, false);
    if (error) return error;

    ApptDB = DmOpenDatabaseByTypeCreator(datebookDBType,
sysFileCDatebook, mode);
    if (! ApptDB) return (1);

    error = ApptAppInfoInit (ApptDB);
    if (error) return error;
}
```

Пример инициализации

```
// Step 3. Get application-specific preferences.  
// Read the preferences / saved-state information. There is  
// only one version of the DateBook preferences so don't  
worry  
// about multiple versions.  
prefsSize = sizeof (DatebookPreferenceType);  
if (PrefGetAppPreferences (sysFileCDatebook, datebookPrefID,  
&prefs, &prefsSize,  
true) != noPreferenceFound)  
{  
    DayStartHour = prefs.dayStartHour;  
    DayEndHour = prefs.dayEndHour;  
    ...  
}
```

Пример инициализации

```
    // Step 4. Initialize any other global variables.  
    TopVisibleAppt = 0;  
    CurrentRecord = noRecordSelected;  
  
    return (error);  
}
```

Другие коды запуска

- Получив код запуска, отличный от `sysAppLaunchCmdNormalLaunch`, приложение проверяет, следует ли его обрабатывать и, если необходимо, вызывает соответствующую реализацию обработчика.

Другие коды запуска

- В большинстве случаев, обрабатывая другие коды запуска, приложение не имеет доступа к глобальным переменным. Обычно глобальные переменные размещаются в памяти только при получении кодов запуска `sysAppLaunchCmdNormalLaunch` и `sysAppLaunchCmdGoto`.
- Если приложение состоит из нескольких сегментов кода, то код за пределами сегмента 0 также недоступен.
- Статические локальные переменные хранятся в динамической куче. Они недоступны, если недоступны глобальные переменные.

Другие коды запуска

- Кроме проверки кода запуска, можно использовать флаги запуска для определения доступности глобальных переменных. В частности, флаг `sysAppLaunchFlagNewGlobals` устанавливается системой, если при этом запуске глобальные переменные доступны.

```
Boolean appHasGlobals = launchFlags & sysAppLaunchFlagNewGlobals;
```


Завершение работы

- Приложение завершает работу, получив событие (не код запуска) `appStopEvent`. Приложение должно распознавать это событие и прерывать свою работу, получив его.
- При завершении приложения следует выполнить такие действия, как сохранение активных записей, закрытие баз данных и сохранение информации о состоянии, которая понадобится при следующем запуске.

Список кодов запуска

- Стандартные коды запуска Palm OS описаны ниже. Все параметры запуска передаются в блоке параметров, в нем же следует размещать результаты работы.

sysAppLaunchCmdAlarmTriggered

- Выполняет быструю обработку будильника (например, назначение нового или выдача звукового оповещения). Этот код запуска посылается как можно ближе к тому моменту, на который установлен будильник. Приложение может выполнить любые быстрые неблокирующие действия.

sysAppLaunchCmdAlarmTriggered

```
typedef struct SysAlarmTriggeredParamType {  
    UInt32 ref;  
    UInt32 alarmSeconds;  
    Boolean purgeAlarm;  
    UInt8 padding;  
} SysAlarmTriggeredParamType;
```

- -> ref Значение устанавливается при создании будильника.
- -> alarmSeconds Дата и время в секундах от 1/1/1904, заданные при создании будильника.
- <- purgeAlarm В процессе обработки установите в true если будильник надо убрать из таблицы будильников.
- padding Не используется.

sysAppLaunchCmdDisplayAlarm

- Выполняет полную, возможно блокирующую обработку будильника. Приложение имеет возможность выполнять длительные или блокирующие действия, например, отобразить сообщение на экране и запросить отклика пользователя.

sysAppLaunchCmdDisplayAlarm

```
typedef struct SysDisplayAlarmParamType {  
    UInt32 ref;  
    UInt32 alarmSeconds;  
    Boolean soundAlarm;  
    UInt8 padding;  
} SysDisplayAlarmParamType;
```

- -> ref Значение устанавливается при создании будильника функцией AlmSetAlarm.
- -> alarmSeconds Дата и время в секундах от 1/1/1904, заданные при создании будильника функцией AlmSetAlarm.
- -> soundAlarm true если следует озвучить событие. В настоящее время не используется.
- padding Не используется.

sysAppLaunchCmdExgReceiveData

- Менеджер обмена (exchange manager) посылает этот код запуска для извещения о необходимости получить данные. Приложение должно, используя функции менеджера обмена, получить данные и сохранить их или произвести другие действия, в зависимости от своих задач.

sysAppLaunchCmdFind

- Этот код запуска используется для реализации глобального поиска. Он посылается системой, когда пользователь вводит текстовую строку в диалог поиска. Система опрашивает все приложения и те из них, которые обрабатывают этот код, могут вернуть записи, соответствующие запросу.
- Многие приложения, которые используют текстовые данные, должны обрабатывать этот код запуска. Получив его, приложение найти все записи, удовлетворяющие запросу и вернуть все совпадения.

sysAppLaunchCmdGoto

- Используется совместно с sysAppLaunchCmdFind и sysAppLaunchCmdExgReceiveData для предоставления пользователю возможности просмотра записи, найденной при глобальном поиске или полученной при передаче данных.
- Приложение должно выполнить большинство действий нормального запуска и затем отобразить запрошенный элемент данных. После этого приложение должно продолжать работу, как и при нормальном запуске.

sysAppLaunchCmdSaveData

- Требуется от приложения сохранить текущие данные, например, перед выполнением глобального поиска.
- Приложение, которое поддерживает команду поиска и может содержать буферизированные данные, должно поддерживать этот код запуска. Как правило, приложение должно производить какие-либо действия, только если оно является активным и содержит буферизированные данные.

sysAppLaunchCmdSyncNotify

- Этот код запуска посылается приложению после выполнения операции HotSync. Он посылается только тем приложениям, чьи базы данных изменились в процессе синхронизации (принадлежность баз данных приложениям определяется по совпадению creator ID).
- Этот код запуска - хорошее средство для обновления, инициализации или проверки новых данных (пересортировка записей, установка будильников и т.п.).

sysAppLaunchCmdSystemReset

- Код запуска для обработки мягкой или жесткой перезагрузки системы. Приложение может выполнить инициализацию, индексирование и другие действия в ответ на перезагрузку системы.

```
typedef struct {  
    Boolean hardReset;  
    Boolean createDefaultDB;  
} SysAppLaunchCmdSystemResetType;
```

- `hardReset` true если произошла жесткая перезагрузка.
- `createDefaultDB` Если true, приложение должно создать исходную базу данных.

sysAppLaunchCmdTimeChange

- Код запуска в ответ на изменение пользователем системного времени. Приложения, которые зависят от текущего времени или даты должны обрабатывать этот код запуска. Например, приложение может отменить некоторые будильники или установить новые.

Цикл обработки событий

- В цикле обработки событий приложение извлекает события из очереди и распределяет их, как правило, события направляются для обработки в системные функции. Например, система знает как обработать клики на форме или меню.
- Обычно приложение остается в цикле обработки событий, пока не получит от системы событие `appStopEvent`. Приложение должно отследить это событие и завершить работу.

Цикл обработки событий

```
static void EventLoop (void)
{
    UInt16 error;
    EventType event;
    do
    {
        EvtGetEvent (&event, evtWaitForever);
        PreprocessEvent (&event);
        if (! SysHandleEvent (&event))
            if (! MenuHandleEvent (NULL, &event, &error))
                if (! ApplicationHandleEvent (&event))
                    FrmDispatchEvent (&event);
    }
    while (event.eType != appStopEvent);
}
```

Цикл обработки событий

- В цикле обработки событий приложение выполняет следующие шаги.
 - Получить событие из очереди. Для этого используется функция

```
void EvtGetEvent (EventType *event, Int32 timeout)
```

- Вызвать PreprocessEvent для перехвата системных кнопок. Не всем приложения требуется функция PreprocessEvent.

Цикл обработки событий

- Вызвать `SysHandleEvent`, чтобы дать системе возможность обработать события (включение/выключение питания, ввод рукописного текста, клики на сенсорном экране, нажатие кнопок, глобальный поиск, оповещение пользователя о разряде батарей). В процессе обработки событий функция `SysHandleEvent` может формировать новые события и помещать их в очередь. Функция `SysHandleEvent` возвращает `true`, если событие было полностью обработано и не требуется никаких дополнительных действий по этому событию. В этом случае приложение может извлечь из очереди следующее событие.

Цикл обработки событий

- Если функция SysHandleEvent не обработала событие, приложение вызывает MenuHandleEvent. Функция возвращает true, если событие было полностью обработано. Эта функция обрабатывает два типа событий:
 - ✓ Пользователь кликнул на области вызова меню, происходит отрисовка меню.
 - ✓ Пользователь кликнул внутри области меню, активизируя пункт меню. MenuHandleEvent удаляет меню с экрана и помещает событие, соответствующее пункту меню, в очередь событий.

Цикл обработки событий

- Если MenuHandleEvent не обработала событие, приложение вызывает функцию ApplicationHandleEvent, реализованную самим приложением. Функция ApplicationHandleEvent обрабатывает только событие frmLoadEvent, она загружает и активизирует форму из ресурсов и устанавливает обработчик событий для активной формы.

Цикл обработки событий

- Если `ApplicationHandleEvent` не обработало событие, приложение вызывает функцию `FrmDispatchEvent`. Эта функция сначала вызывает установленный приложением обработчик событий для активной формы. Таким образом, приложение получает первую возможность обработать события, предназначенные текущей форме. Если этот обработчик возвращает `true`, функция `FrmDispatchEvent` считает событие обработанным и возвращает управление в цикл обработки событий. В противном случае `FrmDispatchEvent` вызывает `FrmHandleEvent` для реализации системной обработки события по умолчанию.

Обработка событий форм

- В процессе обработки событий приложению часто приходится закрывать текущую форму и открывать новую. Это реализуется следующим образом:
 - Приложение вызывает FrmGotoForm для перехода на другую форму. Функция FrmGotoForm помещает событие frmCloseEvent для текущей формы в очередь. Затем в очередь ставятся события frmLoadEvent и frmOpenEvent для новой формы.
 - Получив событие frmCloseEvent, приложение закрывает и уничтожает текущую активную форму.

Обработка событий форм

- Получив frmLoadEvent, приложение загружает и активизирует новую форму. Обычно форма остается активной до своего закрытия. Также устанавливается обработчик событий для формы.
- Получив событие frmOpenEvent, приложение должно провести инициализацию формы и отобразить ее на дисплее.
- После вызова FrmGotoForm все последующие события, проходящие через главный цикл обработки событий в FrmDispatchEvent, передаются в обработчик событий текущей активной формы. Для каждого диалогового окна или формы обработчик событий знает как реагировать на события. Функция FrmHandleEvent обеспечивает основную функциональность интерфейса ПОЛЬЗОВАТЕЛЯ.

События

- События Palm OS являются структурами (определены в файлах Event.h, SysEvent.h, INetMgr.h), которые система передает приложению, когда пользователь взаимодействует с интерфейсом пользователя.
- Структура EventType содержит все данные, ассоциированные с системным событием. Все виды событий имеют некоторые общие данные, многие события имеют специфичные для них данные, заданные объединением в структуре EventType.

События

```
typedef struct {
    eventsEnum eType;
    Boolean penDown;
    UInt8 tapCount;
    Int16 screenX;
    Int16 screenY;
    union{
        ...
    } data;
} EventType;
```


События

- Общие поля этой структуры имеют следующее значение:
 - eType Константа, определяющая тип события.
 - penDown true если стилус был на сенсорном экране в момент события.
 - tapCount Число кликов в заданной области.
 - screenX, screenY Позиция стилуса относительно левого верхнего угла окна.
 - data Специфические для данного события данные, если они заданы. Это поле является объединением, т.е. может рассматриваться как различные структуры в зависимости от значения поля eType.

События

- Некоторые события описаны далее, другие будут рассмотрены совместно с соответствующими элементами системы или интерфейса.

appStopEvent

- При необходимости запустить другое приложение, менеджер событий посылает это событие как запрос на завершение работы приложения. В ответ приложение должно выйти из цикла обработки событий, закрыть используемые базы данных и формы и завершить работу. Если приложение не обрабатывает это событие, система не сможет запустить другое приложение.

keyDownEvent

- Это событие посылается систесой, когда пользователь вводит символ Graffiti, нажимает одну из аппаратных кнопок или нажимает одну из иконок в области ввода, например, Поиск.

```
struct _KeyDownEventType {  
    WChar chr;  
    UInt16 keyCode;  
    UInt16 modifiers;  
};
```

keyDownEvent

- chr Код символа.
- keyCode Не используется.
- modifiers 0 или более следующих значений:
 - ✓ shiftKeyMask Graffiti в режиме case-shift.
 - ✓ capsLockMask Graffiti в режиме cap-shift.
 - ✓ numLockMask Graffiti в цифровом режиме.
 - ✓ commandKeyMask Росчерк доступа к меню или код виртуальной клавиши.
 - ✓ poweredOnKeyMask Нажатие клавиши спровоцировало включение системы.

Формы, окна и диалоги

- Чтобы создать элемент интерфейса пользователя, необходимо создать ресурс, который определяет внешний вид и положение этого элемента. Программно можно взаимодействовать с элементом, как с объектом интерфейса (UI object). Объекты интерфейса в Palm OS являются структурами языка C, которые связаны с одним или более элементами на экране.

Форма

- Форма (form) - область графического интерфейса пользователя для каждого представления приложения. Каждое приложение должно иметь хотя бы одну форму, многие приложения имеют несколько форм. На рисунке показан пример формы.

Edit Memo

Horse of different color
Ruby slippers
Yellow brick road

Done

Форма

- Обычно форма имеет размеры экрана, как показано на рисунке. Другие формы являются модальными диалогами, которые по ширине совпадают с экраном, а по высоте - меньше экрана.

Окно

- Окно (window) определяет область отображения. Эта область может быть на экране или в буфере памяти (off-screen window). Окна в буфере используются для сохранения и восстановления областей экрана. Все формы являются окнами, обратное неверно. Объект окна является частью объекта формы и определяет внешний вид и поведение окна формы. Объект окна определяет координаты и границы отсечения окна.

Формы, окна и диалоги

- Обычно менеджер форм реализует всю отрисовку и обновление по заданным событиям. Однако, для реализации анимации или собственных элементов интерфейса, может понадобиться использовать функции рисования менеджера окна. (В Palm OS форма (form) - аналог окна, а окно (window) - абстрактная область отображения).
- Существует два особых вида форм:
 - Диалог сообщения (Alert Dialogs)
 - Диалог индикации процесса (Progress Dialogs)

Диалог сообщения

- Диалоги сообщения отображаются с использованием функций

```
UInt16 FrmAlert (UInt16 alertId)
```

```
UInt16 FrmCustomAlert (UInt16 alertId, const Char *s1, const Char  
    *s2, const Char *s3)
```

- Получив ID ресурса, менеджер создает и отображает модальный диалог. Когда пользователь кликает одну из кнопок диалога, менеджер уничтожает диалог и возвращает номер элемента выбранной кнопки как результат функции (кнопки нумеруются в порядке отображения, начиная с 0).

Диалог сообщения

- Вторая функция позволяет задать до трех строк, которые заменят переменные ^1, ^2 и ^3 в тексте диалога, заданном ресурсом. Не используйте значение NULL для аргументов s1, s2, и s3. В случае необходимости передавайте строку из одного пробела (" ").

Диалог сообщения

- Существует четыре predetermined типа диалога сообщения:
 - Вопрос (Question)
 - Предупреждение (Warning)
 - Уведомление (Notification)
 - Ошибка (Error)
- Тип диалога определяет отображаемую в диалоге иконку и воспроизводимый при отображении звук.



Функции для работы с формами

```
void FrmGotoForm (UInt16 formId)
```

- Помещает событие frmCloseEvent для текущей формы в очередь. Затем в очередь ставятся события frmLoadEvent и frmOpenEvent для новой формы.

```
void FrmPopupForm (UInt16 formId)
```

- Формирует события frmLoadEvent и frmOpenEvent для заданной формы. Текущая форма не закрывается и к ней можно вернуться с помощью вызова FrmReturnToForm.

```
void FrmReturnToForm (UInt16 formId)
```

- Уничтожает текущую активную форму и делает указанную форму активной.

Функции для работы с формами

```
FormType *FrmInitForm (UInt16 rscID)
```

- Загружает форму из ресурса и инициализирует ее.

```
UInt16 FrmGetActiveFormID (void)
```

```
FormType *FrmGetActiveForm (void)
```

- Возвращает ID или указатель на объект текущей активной формы.

```
void FrmDrawForm (FormType *formP)
```

- Отрисовывает объекты формы. Следует вызывать эту функцию в ответ на событие frmOpenEvent.

```
void FrmSetEventHandler (FormType *formP, FormEventHandlerType  
*handler)
```

- Установить обработчик событий для формы.

```
Boolean FrmDispatchEvent (EventType *eventP)
```

- Передает событие обработчику текущей формы.

Функции для работы с формами

```
void FrmSaveAllForms (void)
```

- Посылает событие frmSaveEvent всем открытым формам.

```
void FrmCloseAllForms (void)
```

- Закрывает все активные формы. Используется при завершении работы приложения.

```
void FrmDeleteForm (FormType *formP)
```

- Удалить форму и освободить память.

```
UInt16 FrmDoDialog (FormType *formP)
```

- Отображает модальный диалог. Возвращает ID нажатой кнопки.

События форм

- Многие события форм обрабатываются системой при вызове `FrmDispatchEvent`, к ним относятся:
 - `frmCloseEvent` Закрывает форму и высвобождает память.
 - `frmTitleEnterEvent` Нажатие стилусом на область заголовка формы.
 - `frmTitleSelectEvent` Клик на области заголовка формы. В Palm OS 3.5 и выше в ответ формируется событие `keyDownEvent` с виртуальным символом `vchrMenu` для отображения меню формы.
 - `frmUpdateEvent` Активизируется перерисовка формы.
- Обработка следующих событий должна быть реализована в приложении.

frmGotoEvent

- Приложение может отправить себе это событие в ответ на код запуска `sysAppLaunchCmdGoto`. Событие сходно с `frmOpenEvent`, требует инициализации и отрисовки формы, но позволяет указать дополнительную информацию для отображения и выделения найденной информации.

frmLoadEvent

- Событие является запросом на загрузку формы в память.

```
struct frmLoad {  
    UInt16 formID;  
} frmLoad;
```

- formID Определенный разработчиком ID формы.

frmOpenEvent

- Событие является запросом на инициализацию и отрисовку формы.

```
struct frmOpen {  
    UInt16 formID;  
} frmOpen;
```

frmSaveEvent

- Запрос на сохранение всех данных в форме.

Пример обработки событий формы

```
#include <PalmOS.h>
#include "hw.h"
static FormPtr gpForm;
```

Пример обработки событий формы

```
Boolean Form1_HandleEvent(EventPtr event)
{
    switch(event->eType)    {
        ...
        case frmOpenEvent:
            gpForm=FrmGetActiveForm( );
            FrmDrawForm(gpForm);
            break;
        case frmCloseEvent:
            FrmEraseForm(gpForm);
            FrmDeleteForm(gpForm);
            break;
        default:
            return false;
    }
    return true;
}
```

Пример обработки событий формы

```
static Boolean ApplicationHandleEvent(EventPtr event)
{
    FormPtr frm;
    Boolean handled=false;

    switch(event->eType)    {
    case frmLoadEvent:
        frm = FrmInitForm (event->data.frmLoad.formID);
        FrmSetActiveForm (frm);
        FrmSetEventHandler(frm , Form1_HandleEvent);
        handled = true;
        break;
    }
    return handled;
}
```


Пример обработки событий формы

```
UInt32 PilotMain(UInt16 launchCode, void *cmdPBP, UInt16
    launchFlags)
{
    EventType event;
    UInt16 error;

    if(launchCode==sysAppLaunchCmdNormalLaunch) {
        FrmGotoForm(Form1);
        do {
            EvtGetEvent(&event,evtWaitForever);
            if(!SysHandleEvent(&event))
                if(!MenuHandleEvent(0,&event,&error))
                    if(!ApplicationHandleEvent(&event))
                        FrmDispatchEvent(&event);
        }
        while(event.eType!=appStopEvent);
        FrmCloseAllForms();
    }
    return 1; }
```

Описание ресурсов приложения

Типы значений в описаниях ресурсов

- .i идентификатор (пример: kFoo)
- .c символ, можно использовать escape-последовательности языка C (пример: "O")
- .s строка, можно использовать escape-последовательности языка C, пример:
"Click Me"
- .ss несколько строк, пример:
"Now is the time for all good " \
"men to come and aid of their country"

Типы значений в описаниях ресурсов

- .n константа или простое арифметическое выражение, можно использовать "+", "-", "*" и "/". Порядок операций - слева направо, можно применять скобки. Выражения вычисляются в целых числах. Примеры:

23

12+3+1

12*(2+3)

'PALM'

Типы значений в описаниях ресурсов

- .р позиция. Координаты могут быть заданы числом, выражением или одним из ключевых слов:
 - AUTO Автоматическая ширина или высота.
 - CENTER Центрировать элемент по гориз. или верт.
 - CENTER@<coord.n> Координата центра элемента.
 - RIGHT@<coord.n> Координата по правой границе.
 - BOTTOM@<coord.n> Координата по нижней границе.
 - PREVLEFT Левая граница предыдущего элемента.
 - PREVRIGHT Правая граница предыдущего элемента.
 - PREVTOP Верхняя граница предыдущего элемента.
 - PREVBOTTOM Нижняя граница пред. элемента.
 - PREVWIDTH Ширина предыдущего элемента.

Синтаксис

- Допускается использовать однострочные комментарии, начинающиеся с `"/"` вне описания элементов. Блочные комментарии ограничиваются `"/"` и `"/"`.
- Допускается подключать внешние файлы директивой `#include`. Подключаемые файлы могут содержать определения констант в следующем виде

```
– .h      #define <Symbol.i><Value.n>

– .inc    <Symbol.i> equ <Value.n>

– .java, .jav      package <PackageName>
public class <ClassName> {
public static final short <Symbol.i> = <Value.n>; }
```

Общие ресурсы приложения

```
APPLICATIONICONNAME ID <AINResourceId.n>  
    [LOCALE <LocaleName.s>]  
    <ApplicationName.s>
```

- Устанавливает имя приложения, которое будет выводиться под иконкой.

```
APPLICATION ID <ApplResourceId.n>  
    [LOCALE <LocaleName.s>]  
    <APPL.s>
```

- Описывает ID создателя приложения, <APPL.s> должно быть ровно 4 символа.

```
VERSION <Version.s>
```

- Задаёт описатель версии программы, например

```
VERSION "1.0 beta"
```

Общие ресурсы приложения

```
STRING ID <StringResourceId.n>  
    [LOCALE <LocaleName.s>]  
    <String.ss>
```

```
STRING ID <StringResourceId.n>  
    [LOCALE <LocaleName.s>]  
FILE <StringFile.s>
```

- Позволяет создать строковый ресурс, например:

```
STRING ID 100 "This is a very long string that shows escape  
    characters \n" \  
        "as well as continued .ss syntax strings"  
STRING ID 101 FILE "string.txt»
```


Общие ресурсы приложения

```
STRINGTABLE ID <StringTableResourceId.n>  
    [LOCALE <LocaleName.s>]  
    <PrefixString.ss>    ... <String.ss>
```

- Определяет набор строк. Пример использования (результат "Units are:Metric"):

```
[source.rcp]
```

```
STRINGTABLE stringTableTypes  
    "Units are:" "Metric" "Imperial"
```

```
[source.c]
```

```
Char string[32];
```

```
SysStringByIndex(stringTableTypes, 0, string, 32);
```

Общие ресурсы приложения

```
ICON [ID <IconResourceId.n>]  
    [LOCALE <LocaleName.s>]  
    <IconFileName.s>
```

- Большая иконка приложения. IconFileName - путь к монохромному файлу WinBMP размерами 32x22, 32x32 или 22x22 пикселя.

Общие ресурсы приложения

```
ICONFAMILY [ID <IconResourceId.n>]  
    <BitmapFileName.s> ... <BitmapFileName.s>  
    [NOCOLORTABLE] [COLORTABLE]  
    [TRANSPARENT r g b] [TRANSPARENTINDEX  
index]
```

- Служит для задания набора иконок под разные цветовые разрешения (<1-bit icon> <2-bit icon> <4-bit icon> <8-bit icon>). [TRANSPARENT <r> <g>] позволяет указать цвет, который будет прозрачным, а [TRANSPARENTINDEX <index>] указать индекс прозрачного цвета в палитре рисунка.

Общие ресурсы приложения

```
SMALLICON [ID <IconResourceId.n>]
```

```
[LOCALE <LocaleName.s>]
```

```
<IconFileName.s>
```

```
SMALLICONFAMILY [ID <IconResourceId.n>]
```

```
<BitmapFileName.s> ... <BitmapFileName.s>
```

```
[NOCOLORTABLE] [COLORTABLE]
```

```
[TRANSPARENT r g b] [TRANSPARENTINDEX  
index]
```

- Аналогично предыдущим, определяют иконки размером 15x9.

Общие ресурсы приложения

BITMAP [<ResType.s>] ID <BitmapResourceId.n>
[LOCALE <LocaleName.s>]
<BitmapFileName.s>
[NOCOMPRESS] [COMPRESS] [FORCECOMPRESS]

BITMAPFAMILY [<ResType.s>]
ID <BitmapResourceId.n>
<BitmapFileName.s> ... <BitmapFileName.s>
[NOCOLORTABLE] [COLORTABLE]
[TRANSPARENT r g b] [TRANSPARENTINDEX index]
[NOCOMPRESS] [COMPRESS] [FORCECOMPRESS]

- Определяют графические ресурсы произвольного размера.

Ресурсы форм

```
FORM ID <FormResourceId.n>  
    AT (<Left.p> <Top.p> <Width.p> <Height.p> )  
    [FRAME] [NOFRAME]  
    [MODAL]  
    [SAVEBEHIND] [NOSAVEBEHIND]  
    [USABLE]  
    [HELPID <HelpId.n>]  
    [DEFAULTBTNID <BtnId.n>]  
    [MENUID <MenuId.n>]  
    [LOCALE <LocaleName.s>]  
BEGIN  
    <OBJECTS>  
  
END
```

Ресурсы форм

- [FRAME | NOFRAME] рисовать или нет рамку вокруг формы
- [MODAL] признак модальной формы (имеет рамку и не принимает событий из-за своих границ)
- [SAVEBEHIND | NOSAVEBEHIND] включает или отключает сохранение области окна перекрываемого другим окном. Использование данной возможности позволяет уменьшить время перерисовки окна, т.к. отрисовывается только перекрытый регион, а не вся форма.

Ресурсы форм

- [USABLE] форма используется, доступна для взаимодействия с пользователем. Этот параметр чаще применяется для элементов формы, совместно с [NONUSABLE]
- [HELPID] ID справки, относящейся к данной форме
- [DEFAULTBTNID] ID кнопки, назначаемой кнопкой по умолчанию
- [MENUID] ID меню формы

Ресурсы форм и диалогов сообщений

```
ALERT ID <AlertResrouceId.n>
    [HELPID <HelpId.n>]
    [DEFAULTBUTTON <ButtonIdx.n>]
    [INFORMATION] [CONFIRMATION] [WARNING]
    [ERROR]
    [LOCALE <LocaleName.s>]
BEGIN
    TITLE <Title.s>
    MESSAGE <Message.ss>
    BUTTONS <Button.s> ... <Button.s>
END
```

Ресурсы форм и диалогов сообщений

- В тексте можно поместить маркеры ^1, ^2, ^3, а в программе можно будет задать текст, который будет выводиться вместо маркера. Пример:

```
ALERT ID 1000
  HELPID 100 DEFAULTBUTTON 1 CONFIRMATION
  LOCALE "enUS"
BEGIN
  TITLE "AlarmHack"
  MESSAGE "Continuing will cause you ^1 years of bad luck\n" \
    "Are you sure?"
  BUTTONS "Ok" "Cancel"
END
```

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

Разработка ПО для портативных компьютеров (КПК)

Palm OS (3)

Жерздев С.В.

Элементы управления

- Элементы управления служат для организации взаимодействия приложения и пользователя в рамках формы.

Метки

- Текстовые метки (label) отображают не редактируемый текст на форме. Используются, например, для отображения текста слева от флажков.

Кнопки

- Кнопки (buttons) отображают текстовую метку в рамке. По умолчанию рамка имеет скругленные углы. Если кнопка не имеет рамки, при ее нажатии инвертируется область вокруг надписи.
- При нажатии стилусом на кнопку, она подсвечивается до отпускания или вывода стилуса за пределы кнопки.



Селектор

- Селектор (selector trigger) отображает текстовую метку в серой прямоугольной рамке. Как правило, используется для отображения модального окна для выбора пользователем некоторого значения (дата, время, шрифт) и последующего отображения этого значения в самом селекторе.



Кнопка с повтором

- Кнопка с повтором (repeating button) похожа на обычную кнопку, но генерирует соответствующие события, пока кнопка нажата, а не только по нажатию/отпусканию.

Кнопки выбора

- Кнопки выбора (push buttons) используются для выбора одного из значений. Из всех кнопок выбора одной группы выделенной может быть только одна.

Priority: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Sort by: ☒ Priority ☐ Due Date

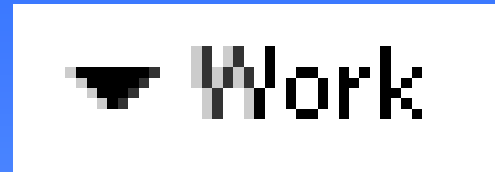
Флажок

- Флажки (check boxes) отображают элемент, который может быть отмечен (checked) или нет (unchecked).
- Флажки, как и кнопки выбора могут объединяться в группы, одновременно в группе может быть отмечен только один элемент.



Списки

- Выпадающий список (popup trigger) отображает текстовую метку и графический элемент, обозначающий вызов всплывающего списка (popup list). Ширина элемента автоматически подстраивается при изменении текстовой метки.



Списки

- Сам по себе этот элемент не содержит никаких данных и является только "заголовком". Данные содержатся в связанном списке (элемент LIST). Связывание элементов POPUPTRIGGER и LIST выполняется посредством элемента POPUPLIST.
- Связанный список обычно имеет признак NONUSABLE. При нажатии на выпадающий список, связанный список становится видимым. Если же список изначально видим (USABLE), то его границы меняются, становясь прямоугольником с закругленными углами и тенью, а после выбора просто восстанавливаются в изначальном виде.

Списки

- Список (list) отображается как вертикальный набор вариантов в рамке. Текущий вариант (элемент списка) подсвечивается. При нажатии на элемент, он подсвечивается, при отпускании в границах элемента он становится текущим выбранным.



Списки

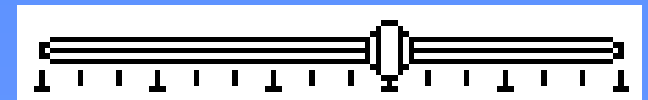
- Если в списке задано больше вариантов, чем может быть отображено одновременно, отображаются маленькие стрелочки (индикаторы прокрутки) на правой границе возле первого и последнего видимых вариантов. При нажатии на них список прокручивается. При скроллинге текущий выбранный элемент не изменяется.

Списки

- Есть два варианта использования списков:
 - Сформировать набор вариантов и предоставить системе обрабатывать события, связанные со списком.
 - Самостоятельно обрабатывать события списка и реализовать его отрисовку на базе динамически формируемых данных.
- Второй вариант позволяет уменьшить расход памяти при работе с большими списками.

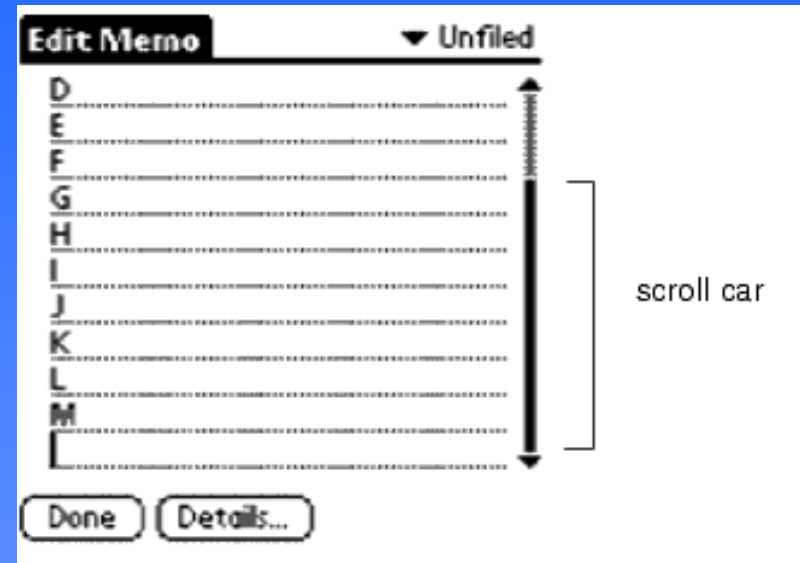
Регуляторы

- Начиная с Palm OS 3.5 поддерживаются регуляторы (slider controls). Регулятор представляет значение в заданном диапазоне.
- Регулятор имеет четыре атрибута: минимальное, максимальное и текущее значение, большой шаг (page jump value), который используется при кликах слева и справа от ползунка.
- Palm OS поддерживает два вида регуляторов: обычные и с обратной связью. Первые генерируют событие при "отпускании" ползунка, вторые - при каждом его перемещении на пиксел.



Полосы прокрутки

- Начиная с Palm OS 2.0 поддерживаются вертикальные полосы прокрутки (scroll bar), которые можно привязать к полям ввода, таблицам, спискам и обрабатывать соответствующие события.



Полосы прокрутки

- Для включения полосы прокрутки в интерфейс надо:
 - Создать соответствующий ресурс, задать ID, положение и размеры полосы прокрутки. Высота должна соответствовать высоте привязанного элемента управления, ширина - 7 пикселей.
 - Задать минимальное и максимальное значение, размер страницы (page size). Минимальное значение обычно 0, максимальное 0 и меняется в программе. Размер страницы определяет шаг прокрутки.
 - Сделать полосу прокрутки частью формы.

Поля ввода

- Поле ввода (field) отображает одну или более строк текста и реализуют:
 - выделение текста
 - прокрутку для многострочных полей
 - вырезание, копирование и вставку
 - выравнивание по левому или правому краю
 - табуляторы
 - режим "только для чтения"
 - режим автоматически увеличиваемой высоты элемента
 - подчеркивание строк
 - ограничение объема текста
 - позиционирование курсора
 - полоса прокрутки

Look Up: Text.....

Поля ввода

- Поля ввода поддерживают символ перевода строки (`\0A`), но не символ возврата каретки (`\0D`). При трансляции файлов ресурсов это учитывается автоматически, но в других случаях - нет.
- Для работы с текстовыми строками следует использовать не стандартные функции C, а функции менеджера строк (string manager). Это делает приложение меньше и устраняет проблемы совместимости (не все стандартные функции C поддерживаются в Palm OS).

События элементов управления

ctlEnterEvent

- Управляющая процедура CtlHandleEvent посылает это событие в ответ на получение penDownEvent в пределах элемента управления.

```
struct ctlEnter {  
    UInt16 controlId;  
    struct ControlType *pControl;  
} ctlEnter;
```

- controlId Определенный разработчиком ID элемента управления.
- pControl Указатель на структуру описания элемента управления (ControlType).

ctlExitEvent

- После события `ctlEnterEvent`, система отслеживает положение стилуса до отпускания. Это событие формируется процедурой `CtlHandleEvent`, если отпускание произошло за пределами элемента управления.

ctlRepeatEvent

- Формируется каждые 0,5 секунды при удержании стилуса на кнопке с повтором и при смещении ползунка хотя бы на один пиксел на регуляторе. Если в ответ на это событие обработчик вернет true, формирование новых событий прекратится.

ctlRepeatEvent

```
struct ctlRepeat {  
    UInt16 controlID;  
    struct ControlType *pControl;  
    UInt32 time;  
    UInt16 value;  
} ctlRepeat;
```

- controlID ID элемента управления.
- pControl Указатель на структуру элемента
- time Время помещения события в очередь
- value Текущее значение для регулятора

ctlSelectEvent

- После события `ctlEnterEvent`, система отслеживает положение стилуса до отпускания. Это событие формируется процедурой `CtlHandleEvent`, если отпускание произошло в пределах элемента управления.

ctlSelectEvent

```
struct ctlSelect {  
    UInt16 controlID;  
    struct ControlType *pControl;  
    Boolean on;  
    UInt8 reserved1;  
    UInt16 value;  
} ctlSelect;
```

- controlID ID элемента управления.
- pControl Указатель на структуру элемента.
- time Время помещения события в очередь.
- on true если элемент управления "отпущен".
- reserved1 Не используется.
- value Текущее значение для регулятора.

lstSelectEvent

- Событие формируется при выборе элемента списка.

```
struct lstSelect {  
    UInt16 listID;  
    struct ListType *pList;  
    Int16 selection;  
} lstSelect;
```

- listID ID списка
- pList Указатель на структуру списка
- selection Номер нового выбранного элемента
 (начиная с 0)

popSelectEvent

- Событие формируется при выборе элемента выпадающего списка.

```
struct popSelect {  
    UInt16 controlID;  
    struct ControlType *controlP;  
    UInt16 listID;  
    struct ListType *listP;  
    Int16 selection;  
    Int16 priorSelection;  
} popSelect;
```

popSelectEvent

- controlID ID ресурса выпадающего списка
- controlP Указатель на структуру выпадающего списка
- listID ID связанного списка
- listP Указатель на структуру связанного списка
- selection Номер нового выбранного элемента (начиная с 0)
- priorSelection Номер предыдущего выбранного элемента (начиная с 0)

sclExitEvent

- Это событие посылается при завершении работы с полосой прокрутки и должно обрабатываться приложением для реализации нединамического режима.

```
struct sclExit {  
    UInt16 scrollBarID;  
    struct ScrollBarType *pScrollBar;  
    Int16 value;  
    Int16 newValue;  
} sclExit;
```

- scrollBarID ID ресурса полосы прокрутки
- pScrollBar Указатель на стр-ру полосы прокрутки
- value Начальное положение полосы прокрутки
- newValue Новое положение полосы прокрутки

sclRepeatEvent

- Это событие посылается в процессе работы с полосой прокрутки и должно обрабатываться приложением для реализации динамического режима.

```
struct sclRepeat {  
    UInt16 scrollBarID;  
    struct ScrollBarType *pScrollBar;  
    Int16 value;  
    Int16 newValue;  
    Int32 time;  
} sclRepeat;
```


sclRepeatEvent

- `scrollBarID` ID ресурса полосы прокрутки
- `pScrollBar` Указатель на структуру полосы прокрутки
- `value` Начальное положение полосы прокрутки
- `newvalue` Новое положение полосы прокрутки
- `time` Системное время создания события

Ресурсы элементов формы

Заголовок формы

```
TITLE <Title.s>
```

Кнопка

```
BUTTON <Label.s> ID <Id.n>  
AT (<Left.p> <Top.p> <Width.p> <Height.p>)  
[USABLE] [NONUSABLE] [DISABLED]  
[LEFTANCHOR] [RIGHTANCHOR]  
[FRAME] [NOFRAME] [BOLDFRAME]  
[FONT <FontId.n>] [GRAPHICAL]  
[BITMAPID <BitmapId.n>] [SELECTEDBITMAPID  
    <BitmapId.n>]
```

- **USABLE** кнопка видима и принимает события. Используется по умолчанию.
- **NONUSABLE** невидима и событий не принимает
- **DISABLED** кнопка видима, но не принимает событий

Кнопка

- LEFTANCHOR | RIGHTANCHOR указывает привязку границы при изменении размеров надписи
- FRAME тонкая овальная граница. Используется по умолчанию.
- NOFRAME границы нет
- BOLDFRAME толстая овальная граница
- RECTFRAME тонкая прямоугольная граница
- FONT <FontID>идентификатор шрифта используемого для вывода надписи на кнопке

Кнопка

- GRAPHICAL признак графической кнопки, для отображения которой используются картинки
- BITMAPID <BitmapID> идентификатор картинки, изображающей кнопку в нормальном состоянии
- SELECTEDBITMAPID <BitmapID>
 идентификатор картинки, изображающей кнопку в нажатом состоянии

Кнопки выбора

```
PUSHBUTTON <Label.s> ID <Id.n>  
AT (<Left.p> <Top.p> <Width.p> <Height.p>)  
[USABLE] [NONUSABLE] [DISABLED]  
[LEFTANCHOR] [RIGHTANCHOR]  
[FONT <FontId>]  
[GROUP <GroupId.n>]  
[GRAPHICAL]  
[BITMAPID <BitmapId.n>] [SELECTEDBITMAPID  
    <BitmapId.n>]
```

Кнопки выбора

- GROUP <GroupID> номер группы. Если не назначен или равен 0, то то кнопка работает как переключаеть принимая одно из двух состояний при нажатии. Если же номер группы задан, то все кнопки группы начинают работать как группа радиокнопок. Т.е. позволяют выбрать одно из нескольких значений.

Повторяющаяся кнопка

```
REPEATBUTTON <Label.s> ID <Id.n>  
AT (<Left.p> <Top.p> <Width.p> <Height.p>)  
[USABLE] [NONUSABLE] [DISABLED]  
[LEFTANCHOR] [RIGHTANCHOR]  
[FRAME] [NOFRAME] [BOLDFRAME]  
[FONT <FontId.n>] [GRAPHICAL]  
[BITMAPID <BitmapId.n>] [SELECTEDBITMAPID  
  <BitmapId.n>]
```

Флажок

```
CHECKBOX <Label.s> ID <Id.n>  
AT (<Left.p> <Top.p> <Width.p> <Height.p>)  
[USABLE] [NONUSABLE] [DISABLED]  
[LEFTANCHOR] [RIGHTANCHOR]  
[FONT <FontId.n>]  
[GROUP <GroupId.n>]  
[CHECKED]
```

- CHECKED признак того, что флажок отмечен

Селектор

```
SELECTORTRIGGER <Label.s> ID <Id.n>  
AT (<Left.p> <Top.p> <Width.p> <Height.p>)  
[USABLE] [NONUSABLE] [DISABLED]  
[LEFTANCHOR] [RIGHTANCHOR]  
[FONT <FontId.n>]  
[GRAPHICAL]  
[BITMAPID <BitmapId.n>] [SELECTEDBITMAPID  
  <BitmapId.n>]
```

Выпадающий список

```
POPUPTRIGGER <Label.s> ID <Id.n>  
AT (<Left.p> <Top.p> <Width.p> <Height.p>)  
[USABLE] [NONUSABLE] [DISABLED]  
[LEFTANCHOR] [RIGHTANCHOR]  
[FONT <FontId.n>]  
[GRAPHICAL]  
[BITMAPID <BitmapId.n>] [SELECTEDBITMAPID  
  <BitmapId.n>]
```

- Сам по себе этот элемент не содержит никаких данных и является только "заголовком". Данные содержатся в связанном списке (LIST). Связывание POPUPTRIGGER и LIST выполняется посредством POPUPLIST.

Выпадающий список

- Связанный список обычно имеет признак NONUSABLE. При нажатии на выпадающий список, связанный список становится видимым, если использовалось NONUSABLE, а после выбора становится невидимым.
- Если же список изначально видим (USABLE), то его границы меняются, становясь прямоугольником с закругленными углами и тенью, а после выбора просто восстанавливаются в изначальном виде, при этом список не убирается с экрана, оставаясь видимым. Сам связанный список может находиться в произвольном месте экрана.

Список

```
LIST <Item.s> ... <Item.s> ID <Id.n>  
AT (<Left.p> <Top.p> <Width.p> <Height.p>)  
[USABLE] [NONUSABLE] [DISABLED]  
[VISIBLEITEMS <NumVisItems.n>]  
[FONT <FontId.n>]
```

- <Item> строка, заключенная в кавычки. Определяет текст элемента списка.
- VISIBLEITEMS количество элементов, которые должны быть видны в списке. Изменяет вертикальный размер списка.

Связка

POPUPLIST ID <ControlId.n> <ListId.n>

- Данный тег выполняет связку между выпадающим списком и обычным списком.
- ControlId идентифицирует элемент POPUPTRIGGER
- ListId идентификатор связанного списка
- Выпадающий список сам по себе не хранит никаких значений и является всего лишь формой отображения обычного списка. Поэтому при работе с выпадающим списком необходимо создавать и обычный список (LIST).

Текстовая метка

```
LABEL <Label.s> ID <Id.n>  
AT ( <Left.p> <Top.p> )  
[USABLE] [NONUSABLE]  
[FONT <FontId.n>]
```


Полоса прокрутки

SCROLLBAR ID <Id.n>

AT (<Left.p> <Top.p> <Width.p> <Height.p>)

[USABLE] [NONUSABLE]

[VALUE <Value.n>] [MIN <MinValue.n>] [MAX
 <MaxValue.n>]

[PAGESIZE <PageSize.n>]

- Полоса прокрутки. Ширина полосы прокрутки всегда равна 7 пикселям.
- VALUE начальное значение
- MIN минимальное значение
- MAX максимальное значение
- PAGESIZE размер страницы.

Регулятор

SLIDER ID <Id.n>

AT (<Left.p> <Top.p> <Width.p> <Height.p>)

[USABLE] [NONUSABLE] [DISABLED]

[VERTICAL]

[FEEDBACK]

[THUMBID <BitmapId.n>] [BACKGROUNDID
 <BitmapId.n>]

[VALUE <Value.n>] [MIN <MinValue.n>] [MAX
 <MaxValue.n>]

[PAGESIZE <PageSize.n>]

Поле ввода

FIELD ID <Id.n>

AT (<Left.p> <Top.p> <Width.p> <Height.p>)

[USABLE] [NONUSABLE] [DISABLED]

[LEFTALIGN] [RIGHTALIGN]

[FONT <FontId.n>]

[EDITABLE] [NONEDITABLE]

[UNDERLINED]

[SINGLELINE] [MULTIPLELINES]

[DYNAMICSIZE] [MAXCHARS <MaxChars.n>]

[AUTOSHIFT] [NUMERIC] [HASSCROLLBAR]

Поле ввода

- EDITABLE | NONEDITABLE доступность редактирования поля. По умолчанию EDITABLE
- UNDERLINED включает подчеркивание поля пунктиром
- SINGLELINE | MULTIPLELINES
 разрешает/запрещает ввод многострочного текста
- DYNAMICSIZE автоматический подбор размера элемента
- MAXCHARS задает максимальную допустимую длину текста

Поле ввода

- AUTOSHIFT управляет автоматическим преобразованием первой буквы предложения в заглавную
- NUMERIC разрешает ввод только цифровых последовательностей. Знаки арифметических операций, в т.ч. "+" и "-" недоступны.
- HASSCROLLBAR управляет видимостью полос прокрутки

ФОН

FORMBITMAP

AT (<Left.p> <Top.p>)

[BITMAP <BitmapId.n>]

[USABLE] [NONUSABLE]

- Выводит картинку на форму в качестве подложки. Поверх могут располагаться другие контролы.
- BITMAP <BitmapId> идентификатор картинки

Таблица

```
TABLE ID <Id.n>
```

```
AT (<Left.p> <Top.p> <Width.p> <Height.p>)
```

```
[ROWS <NumRows.n>] [COLUMNS <NumCols.n>]
```

```
[COLUMNWIDTHS <Col1Width.n> ...  
  <ColNWidth.n>]
```

- Выводит таблицу. Пока таблицы не рассматриваются, т.к. требуют динамического заполнения.
- ROWS число строк таблицы
- COLUMNS число колонок таблицы
- COLUMNWIDTHS значения ширины колонок

Индикатор состояния граффити

GRAFFITISTATEINDICATOR

AT (<Left.p> <Top.p>)

Пример

```
FORM ID 1
    AT (2 2 156 156)
    USABLE MODAL
    HELPID 1 MENUID 1
    LOCALE "enUS"
BEGIN
    TITLE "AlarmHack"

    LABEL "Repeat Datebook alarm sound" AUTOID AT (CENTER 16)
    PUSHBUTTON "1" ID 2001 AT (20 PrevBottom+2 12) AUTO GROUP 1
    PUSHBUTTON "2" ID 2002 AT (PrevRight+1 PrevTop PrevWidth PrevHeight)
        GROUP 1

    LABEL "Alarm sound:" AUTOID AT (24 PrevBottom+4)
    POPUPTRIGGER "" ID 5000 AT (PrevRight+4 PrevTop 62 AUTO) LEFTANCHOR
    LIST "Standard" "Bleep" ID 6000 AT (PrevLeft PrevTop 52 1) VISIBLEITEMS
        2 NONUSABLE
    POPUPLIST ID 5000 6000

    GRAFFITISTATEINDICATOR AT (100 100)
END
```

Функции элементов управления

- Для работы с заголовком формы используются функции

```
const Char *FrmGetTitle (const FormType *formP)
```

```
void FrmSetTitle (FormType *formP, Char *newTitle)
```

- Отобразить текст справки по ID ресурса строки
МОЖНО С ПОМОЩЬЮ

```
void FrmHelp (UInt16 helpMsgId)
```

Функции элементов управления

- Для работы с элементами управления используются структуры, описывающие элементы формы. Получить указатель на структуру элемента управления можно в параметрах события или с помощью функции

```
void *FrmGetObjectPtr (const FormType *formP, UInt16 objIndex)
```

- по указателю на структуру формы и номеру объекта (не ID!). Номер объекта определяется с помощью

```
UInt16 FrmGetObjectIndex (const FormType *formP, UInt16 objID)
```

Функции элементов управления

- Далее можно работать с указателем на структуру объекта, передавая его в функции чтения и установки параметров элемента управления

```
Boolean CtlEnabled (const ControlType *controlP)
const Char *CtlGetLabel (const ControlType *controlP)
Int16 CtlGetValue (const ControlType *controlP)
void CtlSetEnabled (ControlType *controlP, Boolean enable)
void CtlSetLabel (ControlType *controlP, const Char *newLabel)
void CtlSetValue (ControlType *controlP, Int16 newValue)
```

Функции элементов управления

- Регуляторы и полосы прокрутки имеют специфичные функции

```
void CtlGetSliderValues (const ControlType *ctlP, UInt16  
    *minValueP, UInt16 *maxValueP, UInt16 *pageSizeP, UInt16  
    *valueP)
```

```
void CtlSetSliderValues (ControlType *ctlP, const UInt16  
    *minValueP, const UInt16 *maxValueP, const UInt16 *pageSizeP,  
    const UInt16 *valueP)
```

```
void Sc1GetScrollBar (const ScrollBarPtr bar, Int16 *valueP,  
    Int16 *minP, Int16 *maxP, Int16 *pageSizeP)
```

```
void Sc1SetScrollBar (const ScrollBarPtr bar, Int16 value, const  
    Int16 min, const Int16 max, const Int16 pageSize)
```

Функции элементов управления

- Для работы с полями ввода используются следующие функции

```
MemHandle FldGetTextHandle (const FieldType* fldP)  
Char* FldGetTextPtr (FieldType* fldP)
```

- Возвращают дескриптор и указатель на область памяти с текстом. Работа с памятью будет рассмотрена позже.

```
void FldSetText (FieldType* fldP, MemHandle textHandle, UInt16  
    offset, UInt16 size)  
void FldSetTextHandle (FieldType* fldP, MemHandle textHandle)  
void FldSetTextPtr (FieldType* fldP, Char* textP)
```

- Привязка поля ввода к дескриптору или указателю на область памяти. Последняя функция не может использоваться с редактируемыми полями.

Функции элементов управления

- После изменения текста необходимо перерисовать поле ввода с помощью

```
void FldDrawField (FieldType* fldP)
```

- Для работы со списками применяется следующий набор функций

```
void LstDrawList (ListType *listP)
```

```
Int16 LstGetNumberOfItems (const ListType *listP)
```

```
Int16 LstGetSelection (const ListType *listP)
```

```
Char * LstGetSelectionText (const ListType *listP, Int16 itemNum)
```

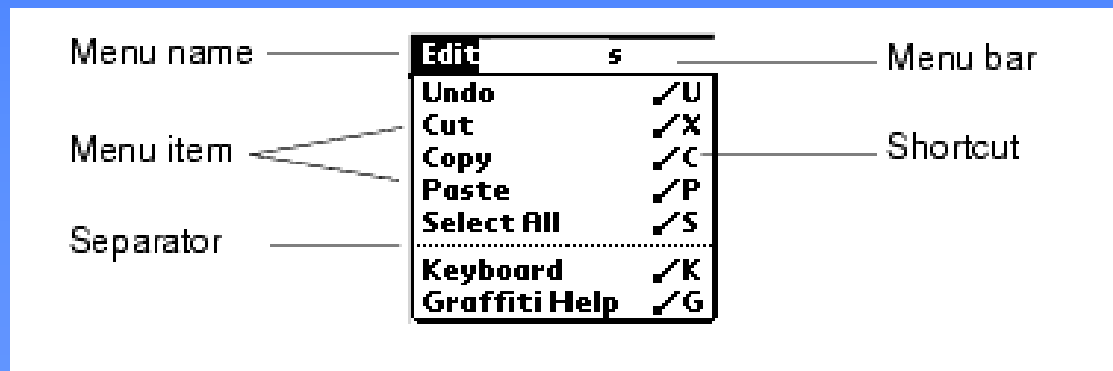
```
void LstMakeItemVisible (ListType *listP, Int16 itemNum)
```

```
void LstSetSelection (ListType *listP, Int16 itemNum)
```

```
void LstSetTopItem (ListType *listP, Int16 itemNum)
```

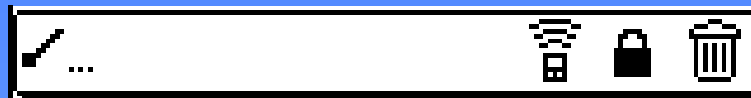
Меню

- Область меню (menu bar) отображается при нажатии на иконку меню или заголовок формы. Область меню - горизонтальный список названий меню (menu title), появляющийся на верхней границе экрана, над всеми окнами приложения. Нажатие на название меню подсвечивает его и отображает само меню (menu) под его названием.



Панель команд

- Некоторые пункты меню могут быть в виде кнопок вынесены на панель команд, появляющуюся снизу экрана по росчерку команды.



Ресурсы меню

```
MENU ID <MenuResourceId.n>
```

```
[ LOCALE <LocaleName.s> ]
```

```
BEGIN
```

```
    <PULLDOWNS>
```

```
END
```

- Описывает область меню, включая все меню.

<PULLDOWNS> - одна или более записей,
описывающих меню, вида:

```
PULLDOWN <PullDownTitle.s>
```

```
BEGIN
```

```
    <MENUITEMS>
```

```
END
```

Ресурсы меню

- В свою очередь пункты меню <MENUITEMS> описываются следующим образом:

```
MENUITEM <MenuItem.s> ID <MenuItemId.n>  
[AccelChar.c]
```

- Кроме названия, для пункта меню можно задать клавишу (росчерк) быстрого доступа.

```
MENUITEM SEPARATOR
```

- Пункты можно разделять сепараторами.

Пример

```
MENU ID 100
    LOCALE "enUS"
BEGIN
    PULLDOWN "File"
    BEGIN
        MENUITEM "Open..." ID 100 "O"
        MENUITEM SEPARATOR
        MENUITEM "Close..." ID 101 "C"
    END

    PULLDOWN "Options"
    BEGIN
        MENUITEM "Get Info..." ID 200 "I"
    END
END
```

События меню

menuEvent

- Событие формируется при выборе пункта меню одним из способов:
 - Нажатие на него в выпадающем меню
 - Использование рошчерка быстрого доступа
 - Нажатие одной из кнопок панели команд

```
struct menu {  
    UInt16 itemID;  
} menu;
```

- itemID ID выбранного пункта меню.

menuCmdBarOpenEvent

- Это событие формируется при отображении командной панели. Приложение может добавить свои элементы на эту панель с использованием функции MenuCmdBarAddButton.
- Возвратив true при обработке этого события, приложение предотвратит отображение командной панели.

Функции для работы с меню

```
Err MenuCmdBarAddButton (UInt8 where, UInt16 bitmapId,  
MenuCmdBarResultType resultType, UInt32 result, Char *nameP)
```

- Помещает новую кнопку на командную панель
 - ->where Позиция новой кнопки (справа налево с 1) или одна из констант menuCmdBarOnLeft (рекомендуется), menuCmdBarOnRight
 - ->bitmapId ID картинки для кнопки (реком. 16*13)
 - ->resultType Тип данных в следующем поле. Как правило, menuCmdBarResultMenuItem
 - ->result Данные, передаваемые при нажатии кнопки. Как правило, ID соответствующего пункта меню
 - ->nameP Указатель на текст для отображения в качестве статусного сообщения. Если NULL, используется текст соответствующего пункта меню

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

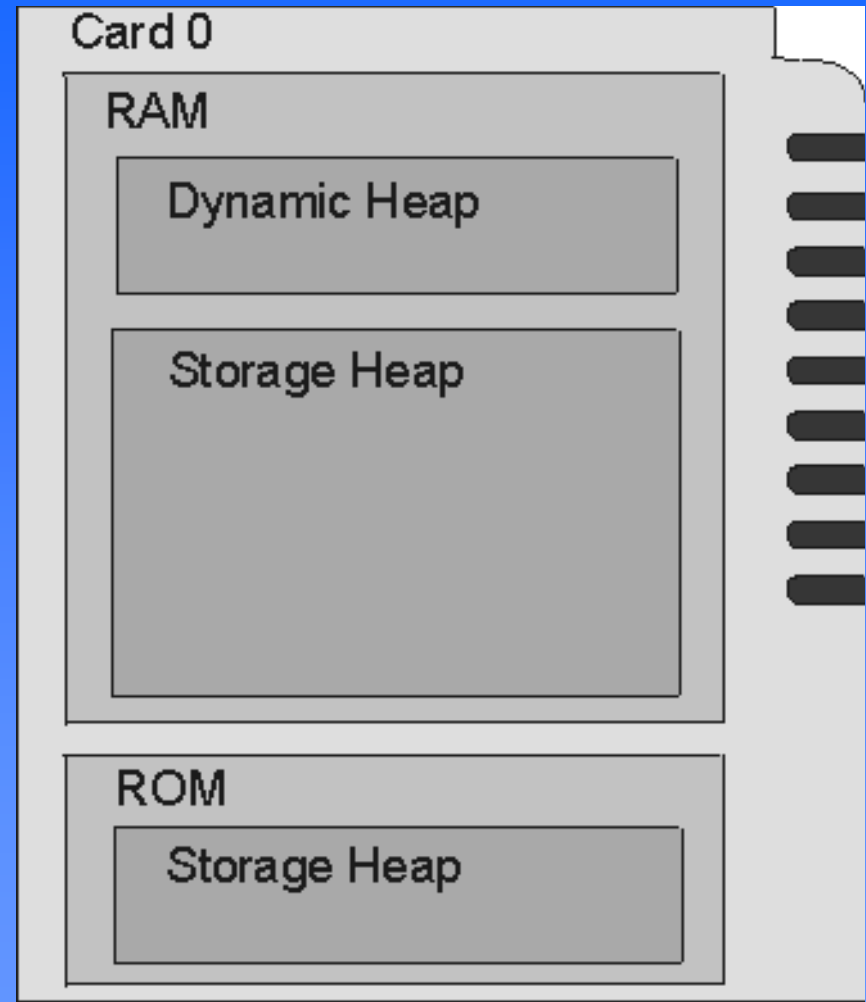
Разработка ПО для портативных компьютеров (КПК)

Palm OS (4)

Жерздев С.В.

Управление памятью

- Физически память расположена на картах (Card) памяти, которые нумеруются - 0, 1, Каждая карта памяти может иметь RAM и ROM сегменты (RAM - random access memory, ROM - read only memory).



Управление памятью

- PalmOS делит все пространство RAM сегмента на Dynamic Heap и Storage Heap. Storage Heap это эквивалент диска (HDD), там находятся только файлы. В Dynamic Heap находятся все динамические объекты приложения, операционной системы, библиотек, модулей, а также стек. В PalmOS код приложения не загружается в Dynamic Heap для выполнения, выполнение происходит по месту (inplace), т.е. все программы, как и сама PalmOS, всегда выполняются прямо из Storage Heap. Это прозрачно для приложений.

Управление памятью

- Область Storage Heap управляется менеджером баз данных (Database Manager). Динамическая память управляется с помощью функций менеджера памяти (Memory Manager).
- Динамическая память используется для размещения
 - глобальных переменных ОС,
 - динамически выделяемых областей ОС,
 - глобальных и статических переменных приложения,
 - стека приложения,
 - динамически выделяемых областей приложения.

Управление памятью

- В Palm OS все данные располагаются в областях (chunk) менеджера памяти. Область - непрерывный фрагмент памяти размером от 1 байта до чуть менее 64 KB, который может быть размещен с помощью менеджера памяти Palm OS.
- Приложение может размещать в динамической куче как непереключаемые (nonrelocatable) области (называемые указателями, pointers), так и переключаемые области (называемые дескрипторами, handles).

Управление памятью

- Для размещения перемещаемой области используется функция

```
VoidHand myHandle = MemHandleNew(chunkSize)
```

- Нельзя размещать область нулевого размера.
- Для чтения или записи в перемещаемый блок, его необходимо временно блокировать.
Незаблокированные блоки могут быть перемещены при запросе на выделение памяти, указатели на них недействительны.

```
void *myPointer = MemHandleLock(myHandle);  
// do something with myPointer  
MemHandleUnlock(myHandle);
```

Управление памятью

- Вызовы MemHandleLock и MemHandleUnlock могут быть вложенными, поскольку MemHandleLock увеличивает счетчик блокировок (до 15 одновременных блокировок на область). При попытке превысить значение счетчика или количество разблокирований, будет выдано сообщение об ошибке.
- Область может быть разблокирована не только по дескриптору, но и по указателю с помощью функции

```
Err MemPtrUnlock (MemPtr p)
```

Управление памятью

- Можно получить дескриптор перемещаемой области по ее указателю

```
MemHandle MemPtrRecoverHandle (MemPtr p)
```

- Для размещения неподвижной области используется функция

```
struct s *myS = MemPtrNew(sizeof(struct s));
```

- Для освобождения областей (в том числе заблокированных) используются функции

```
MemPtrFree(myPointer);
```

```
MemHandleFree(myHandle);
```


Управление памятью

- При размещении области, ей присваивается ID владельца. При завершении работы приложения, менеджер памяти освобождает все области памяти, принадлежащие приложению.
- Для сохранения области (например, для передачи ее другому приложению) можно установить ID владельца в 0 с помощью вызова функций

```
Err MemHandleSetOwner (MemHandle h, UInt16 owner)
```

```
Err MemPtrSetOwner (MemPtr p, UInt16 owner)
```

Управление памятью

- сравнить две области памяти

```
Int16 MemCmp (const void* s1, const void* s2, Int32 numBytes)
```

- определить размер области

```
UInt32 MemHandleSize (MemHandle h)
```

```
UInt32 MemPtrSize (MemPtr p)
```

- изменить размер области (сохраняя ее положение)

```
Err MemHandleResize (MemHandle h, UInt32 newSize)
```

```
Err MemPtrResize (MemPtr p, UInt32 newSize)
```

- определить объем доступной памяти и
максимальный размер области для кучи
(динамическая куча имеет идентификатор 0)

```
Err MemHeapFreeBytes (UInt16 heapID, UInt32* freeP, UInt32* maxP)
```

Управление памятью

- переместить данные в другую область

```
Err MemMove (void* dstP, const void* sP, Int32 numBytes)
```

- заполнить область некоторым значением

```
Err MemSet (void* dstP, Int32 numBytes, UInt8 value)
```

- Все функции из серии MemPtrXxx могут применяться и к перемещаемым областям (пока они заблокированы).

Символы и строки

- Для работы с текстовыми строками в Palm OS предусмотрен менеджер строк (string manager). Начиная с версии 3.1, ОС содержит набор функций для работы с национальными символами - менеджер текстов (text manager).

СИМВОЛЫ

- В зависимости от локализации ОС, для хранения символов могут использоваться как одно-, так и многобайтовые коды. Для корректной работы приложение должно пользоваться функциями менеджера текстов и не делать предположений о размере символов.
- Определяйте символьные переменные, используя тип `WChar`:

```
WChar ch; // Right. 16-bit character.
```

```
Char ch; // Wrong. 8-bit character.
```

- Хотя строки и определяются как `Char *`, они могут использовать многобайтную кодировку символов.

СИМВОЛЫ

- В нескольких заголовочных файлах определены символьные константы. Так, в файле Chars.h содержатся символы, обязательно представленные на любом устройстве, независимо от текущей кодировки. Другие заголовочные файлы содержат символы, специфичные для конкретных кодировок. Использование символьных констант предпочтительнее использования непосредственных значений, например:

```
WChar ch = 'a'; // WRONG! Don't use.  
WChar ch = chrSmall_A_RingAbove;
```

- Количество байт в строке для хранения символа
`UInt16 TxtCharSize (WChar inChar)`

СИМВОЛЫ

- В менеджере текстов определены следующие функции для работы с символами.

- Получение атрибутов символа

`UInt16 TxtCharAttr (WChar inChar)`

- Возвращаемое значение - комбинация флагов
 - `charAttrSpace` Пробел, табуляция или перевод строки
 - `charAttrAlNum` Алфавитно-цифровой
 - `charAttrAlpha` Алфавитный
 - `charAttrCntrl` Управляющий символ
 - `charAttrGraph` Отображаемый символ (не пробельный, не управляющий или виртуальный)
 - `charAttrDelim` Разделитель слов (пробельный или знак препинания)

СИМВОЛЫ

- Обычно для определения конкретного атрибута используются следующие макросы:

`TxtCharIsAlNum (ch)`

`TxtCharIsAlpha (ch)`

`TxtCharIsCntrl (ch)`

`TxtCharIsDelim (ch)`

`TxtCharIsDigit (ch)`

`TxtCharIsGraph (ch)`

`TxtCharIsHardKey (m, ch)`

`TxtCharIsHex (ch)`

`TxtCharIsLower (ch)`

`TxtCharIsPrint (ch)`

`TxtCharIsPunct (ch)`

`TxtCharIsSpace (ch)`

`TxtCharIsUpper (ch)`

СИМВОЛЫ

- Если символ не содержится в текущем шрифте, на его месте отображается пустой (для однобайтовых кодировок) или закрашенный (для двухбайтовых кодировок) прямоугольник. Чаще всего в таком случае необходимо использовать в приложении другой шрифт, иногда это означает использование кода, которому не соответствует никакой символ. Проверить код на корректность можно с помощью функции

```
Boolean TxtCharIsValid (WChar inChar)
```

СИМВОЛЫ

- В качестве параметра события могут передаваться виртуальные символы (virtual characters), которые соответствуют нажатиям на аппаратные кнопки устройства и другим системным событиям:

```
if (TxtGlueCharIsVirtual
    (eventP->data.keyDown.modifiers,
    eventP->data.keyDown.chr)) {
    if (TxtCharIsHardKey
        (event->data.keyDown.modifiers,
        event->data.keyDown.chr)) {
        // Handle hard key virtual character.
    } else {
        // Handle standard virtual character.
    }
} else {
    // Handle regular character.
}
```

СИМВОЛЫ

- Получить текущую кодировку для устройства
МОЖНО ИЗ СИСТЕМНЫХ НАСТРОЕК:

```
UInt32 encoding;  
Char*  encodingName;  
if (FtrGet(sysFtrCreator, sysFtrNumEncoding,  &encoding) != 0)  
encoding = charEncodingPalmLatin;  
    //default encoding  
if (encoding == charEncodingPalmSJIS) {  
    // encoding for Palm Shift-JIS  
} else if (encoding == charEncodingPalmLatin) {  
    // extension of ISO Latin 1  
}  
  
// The following text manager function returns the  
// official name of the encoding as required by  
// Internet applications.  
encodingName = TxtEncodingName(encoding);
```

Строки

- Строки могут содержать символы длиной от 1 до 4 байт каждый. Хотя символьные переменные имеют размер 2 байта, при добавлении символов к строке они могут уменьшаться до 1 байта. Таким образом, строка может содержать смесь символов разного размера. Для работы с такими строками используются функции менеджера текстов. Функции менеджера строк также могут использоваться, но они оперируют размером строки в байтах, а не в символах.
- Все функции, которые возвращают длину строки (FldGetTextLength или StrLen), возвращают размер строки в байтах, а не количество символов.

Строки

- При работе с указателями на символы требуется быть аккуратным, чтобы не получить указатель в середину многобайтового символа. Вместо традиционного увеличения или уменьшения указателя на некоторую величину следует использовать следующие функции:

```
WChar TxtGetChar (const Char* inText, UInt32 inOffset)
UInt16 TxtGetNextChar (const Char* inText, UInt32 inOffset,
    WChar* outChar)
UInt16 TxtGetPreviousChar (const Char* inText, UInt32 inOffset,
    WChar* outChar)
TxtNextCharSize (inText, inOffset)
TxtPreviousCharSize (inText, inOffset)
UInt16 TxtSetNextChar (Char* ioText, UInt32 inOffset, WChar
    inChar)
```

Строки

- Например:

```
UInt16 i = 0;
while (i < bufferLength) {
    i += TxtGetNextChar(buffer, i, &ch);
    //do something with ch.
}
```

- или для получения символа, содержащего заданный байт (в примере - последний байт строки):

```
TxtCharBounds (buffer, bufferLength - 1, &start, &end);
i = start;
while (i > 0) {
    i -= TxtGetPreviousChar(buffer, i, &ch);
    //do something with ch.
}
```

Строки

- Для сравнения строк используются функции (корректно обрабатываются разные представления одного символа)

```
Int16 TxtCompare (const Char* s1, UInt16 s1Len, UInt16*  
    s1MatchLen, const Char* s2, UInt16 s2Len, UInt16* s2MatchLen)  
Int16 TxtCaselessCompare (const Char* s1, UInt16 s1Len, UInt16*  
    s1MatchLen, const Char* s2, UInt16 s2Len, UInt16* s2MatchLen)
```

- -> s1 Указатель на первый буфер текста. Не NULL.
- -> s1Len Длина текста в байтах.
- <- s1MatchLen Длина в байтах совпадающей части в первом тексте.
- -> s2 Указатель на второй буфер текста. Не NULL.
- -> s2Len Длина второго текста.
- <- s2MatchLen Длина в байтах совпадающей части во втором тексте.

Строки

- Возвращается значение
 - < 0 Если $s1$ предшествует $s2$ в алфавитном порядке.
 - > 0 Если $s1$ следует за $s2$ в алфавитном порядке.
 - 0 Если строки совпадают.
- Функции `StrCompare` и `StrCaselessCompare` имеют тот же эффект, но не возвращают длину совпадающих фрагментов.

Строки

- Особый случай сравнения - поиск подстроки. Для этого используется функция

```
Boolean TxtFindString (const Char* inSourceStr, const Char*  
inTargetStr, UInt32* outPos, UInt16* outLength)
```

- -> inSourceStr Указатель на строку. Не должно быть NULL.
- -> inTargetStr Указатель на искомую подстроку.
- <- outPos Указатель на смещение совпадения в inSourceStr.
- <- outLength Указатель на длину совпадения в байтах.

Динамическое формирование строк

- При разработке локализованных приложений строки сохраняются в ресурсах. Если содержимое строки должно формироваться динамически, в ресурсах сохраняются шаблоны строк, в которые подставляются нужные значения.
- Для этого используется функция менеджера строк

```
Char* TxtParamString (const Char* inTemplate, const Char* param0,  
    const Char* param1, const Char* param2, const Char* param3)
```
- Эта функция заменяет последовательности ^0, ^1, до ^3 на соответствующие параметры.

Динамическое формирование строк

- Если требуется использовать больше параметров (от 0 до 9) можно воспользоваться функцией

```
UInt16 TxtReplaceStr (Char* ioStr, UInt16 inMaxLen, const Char*  
    inParamStr, UInt16 inParamNum)
```

- Пример применения шаблона:

```
static void EditViewSetTitle (void)  
{  
    Char* titleTemplateP;  
    FormPtr frm;  
    Char posStr [maxStrIToALen];  
    Char totalStr [maxStrIToALen];  
    UInt16 pos;  
    UInt16 length;
```

Динамическое формирование строк

```
// Format as strings, the memo's position within  
// its category, and the total number of memos  
// in the category.
```

```
pos = DmPositionInCategory (MemoPadDB,  
CurrentRecord, RecordCategory);  
StrIToA (posStr, pos+1);
```

```
if (MemosInCategory == memosInCategoryUnknown)  
MemosInCategory = DmNumRecordsInCategory  
(MemoPadDB, RecordCategory);  
StrIToA (totalStr, MemosInCategory);
```

Динамическое формирование строк

```
// Get the title template string. It contains
// '^0' and '^1' chars which we replace with the
// position of CurrentRecord within
// CurrentCategory and with the total count of
// records in CurrentCategory ().
titleTemplateP = MemHandleLock (DmGetResource
(strRsc, EditViewTitleTemplateStringString));

EditViewTitlePtr = TxtGlueParamString(titleTemplateP, posStr,
totalStr, NULL, NULL);

// Now set the title to use the new title
// string.
frm = FrmGetFormPtr (MemoPadEditForm);
FrmSetTitle (frm, EditViewTitlePtr);
MemPtrUnlock(titleTemplateP);
}
```

Другие функции

- для манипуляции строками

```
Int16 StrCaselessCompare (const Char* s1, const Char* s2)
```

```
Int16 StrCompare (const Char* s1, const Char* s2)
```

```
Char* StrCat (Char* dst, const Char* src)
```

```
Char* StrCopy (Char* dst, const Char* src)
```

```
UInt16 StrLen (const Char* src)
```

- ПОИСКА

```
Char* StrChr (const Char* str, WChar chr)
```

```
Char* StrStr (const Char* str, const Char* token)
```

- и преобразования

```
Int32 StrAToI (const Char* str)
```

```
Char* StrIToA (Char* s, Int32 i)
```

```
Char* StrIToH (Char* s, UInt32 i)
```

```
Char* StrToLower (Char* dst, const Char* src)
```

```
Char* StrLocalizeNumber (Char* s, Char thousSep, Char decSep)
```

```
Char* StrDelocalizeNumber (Char* s, Char thousSep, Char decSep)
```

Базы данных

- Каждая область памяти, предназначенная для длительного хранения данных является записью (record) в базе данных, реализуемой менеджером данных. В Palm OS база данных (database) - просто список областей памяти и связанный с ним заголовок базы данных. Как правило, элементы базы данных логически объединяются по некоторому признаку, например, база данных может содержать все записи телефонной книги.
- Хранение данных в базе данных согласовано с принципами менеджера памяти. Каждая запись в базе является областью (chunk) менеджера памяти.

Базы данных

- База данных в целом имеет следующие атрибуты:
 - имя базы (до 32 байт)
 - тип и создатель базы (по 4 байта)
 - флаги, включая
 - ✓ требуется сохранение базы при синхронизации
 - ✓ защита от копирования
 - ✓ скрытая база
 - ✓ "запускаемая" база (приложение)
 - ✓ только для чтения
 - ✓ база ресурсов
 - ✓ файловый поток
 - номер версии базы, определяемый приложением
 - счетчик модификаций
 - информация приложения

Базы данных

- Каждая запись имеет следующие атрибуты:
 - уникальный идентификатор записи, сохраняется при модификациях
 - номер определяемой пользователем категории записи от 0 до 15 (0 - категория не определена)
 - флаги
 - ✓ удаленная запись, используется при синхронизации (данные могут присутствовать или нет)
 - ✓ запись модифицирована
 - ✓ запись используется для чтения или записи
 - ✓ секретная запись

Управление базами данных

- База данных создается вызовом

```
Err DmCreateDatabase (UInt16 cardNo, const Char * nameP, UInt32  
    creator, UInt32 type, Boolean resDB)
```

- Номер карты, как правило, устанавливается в 0 (встроенная память). Имя базы должно быть уникально и обычно дополняется кодом создателя, например

```
Database1-Neil
```

- Последний параметр указывает, что база содержит ресурсы.

Управление базами данных

- Типичный пример создания базы

```
// Find the Customer database. If it doesn't exist, create it.
gDB = DmOpenDatabaseByTypeCreator(kCustType, kSalesCreator,
    mode);
if (! gDB) {
    err = DmCreateDatabase(0, kCustName, kSalesCreator, kCustType,
        false);
    if (err)
        return err;
    gDB = DmOpenDatabaseByTypeCreator(kCustType, kSalesCreator,
        mode);
    if (!gDB)
        return DmGetLastError();
    // code to initialize records and application info
}
```

Управление базами данных

- Открытие базы обычно производится по ее типу и создателю:

```
DmOpenRef DmOpenDatabaseByTypeCreator (UInt32 type, UInt32 creator, UInt16 mode)
```

- Режим открытия обычно dmModeReadWrite или dmModeReadOnly. Для доступа к секретным записям используется флаг dmModeShowSecret.

Пример:

```
SystemPreferencesType sysPrefs;  
// Determine if secret records should be shown.  
PrefGetPreferences(&sysPrefs);  
if (!sysPrefs.hideSecretRecords)  
    mode |= dmModeShowSecret;  
gDB = DmOpenDatabaseByTypeCreator(kCustType, kSalesCreator, mode);
```

Управление базами данных

- Если приложение использует несколько баз, можно осуществить поиск баз данных по имени или по типу/создателю.

- После работы база закрывается

```
Err DmCloseDatabase (DmOpenRef dbP)
```

- Иногда перед закрытием требуется освободить заблокированные записи

```
err = DmResetRecordStates(gDB);
```

```
err = DmCloseDatabase(gDB);
```

Работа с записями

- Создание новой записи в базе данных производится вызовом

```
myRecordHandle = DmNewRecord(gDB, &recordIndex, recordSize)
```

- В качестве параметров уазываются открытая база, желаемый индекс записи и начальный размер. Размер записи может быть затем изменен с помощью MemHandleSetSize, как и для любой области памяти. В параметре recordIndex возвращается реальный индекс мозданной записи.
- Индексы записей образуют непрерывную последовательность с 0. При указании индекса в этом диапазоне, индексы записей смещаются для организации вставки.

Работа с записями

- Вставка записи в начало базы

```
UInt recordIndex = 0;  
myRecordHandle = DmNewRecord(gDB, &recordIndex, recordSize);
```

- Добавление записи в конец базы

```
UInt recordIndex = dmMaxRecordIndex;  
myRecordHandle = DmNewRecord(gDB, &recordIndex, recordSize)  
// now recordIndex contains the actual index
```

Работа с записями

- Существует возможность определить функцию сравнения записей для организации их в некотором порядке. Указав ее в качестве параметра DmQuickSort или DmInsertionSort, можно упорядочить записи в базе данных. При этом с помощью DmFindSortPosition можно определить позицию для вставки новой записи и далее пользоваться бинарным поиском.

Работа с записями

- Запись в базе данных может быть найдена по уникальному идентификатору

```
UInt recordNumber;
```

```
err = DmFindRecordByID(gDB, uniqueID, &recordNumber);
```

- Кроме того, если записи упорядочены по некоторому ключу, можно организовать бинарный поиск.

Работа с записями

- Кроме поиска по индексу, можно реализовать последовательный перебор записей заданной категории с игнорированием, если установлено, секретных записей.

```
UInt theCat = dmAllCategories; // could be a specific
UInt totalItems = DmNumRecordsInCategory(gDB, theCat);
UInt i;
UInt recordNum = 0;
for (i = 0; i < totalItems; i++) {
VoidHand recordH = DmQueryNextInCategory(gDB, &recordNum,
    theCat);
// at this point, recordNum contains the desired record number.
// You could use DmGetRecord to get write-access, and then
// DmReleaseRecord when finished
// do something with recordH
}
```

Работа с записями

- Чтение и установка уникального идентификатора и атрибутов записи, в том числе категории и секретности осуществляется следующим образом

```
ControlPtr privateCheckbox;  
UInt attributes;  
Boolean isSecret;  
UInt32 uniqueID;  
  
DmRecordInfo(CustomerDB, recordNumber, &attributes, &uniqueID,  
    NULL);  
isSecret = (attributes & dmRecAttrSecret) == dmRecAttrSecret;  
...  
attributes |= dmRecAttrSecret;  
// attributes &= ~dmRecAttrSecret;  
DmSetRecordInfo(CustomerDB, recordNumber, &attributes, NULL);
```

Работа с записями

- **Функция**

```
Err DmSetRecordInfo (DmOpenRef dbP, UInt16 index, UInt16* attrP,  
    UInt32* uniqueIDP)
```

- **позволяет также изменить уникальный идентификатор записи.**
- **Чтение записей осуществляется с помощью блокировки соответствующей области памяти**

```
VoidHand myRecord = DmQueryRecord(gDB, recordNumber);  
StructType *s = MemHandleLock(myRecord);  
DoSomethingReadOnly(s->field);  
MemHandleUnlock(myRecord);
```

Работа с записями

- Для изменения записи кроме блокировки области памяти необходимо пометить запись как занятую, а затем снять эту пометку. Кроме того, нельзя писать непосредственно в область памяти соответствующую записи, необходимо использовать функцию DmWrite

```
VoidHand myRecord = DmGetRecord(gDB, recordNumber);  
StructType *s = MemHandleLock(myRecord);  
StructType theStructure;  
theStructure = *s;  
theStructure.field = newValue;  
DmWrite(gDB, s, 0, &theStructure, sizeof(theStructure));  
MemHandleUnlock(myRecord);  
DmReleaseRecord(gDB, recordNumber, true);
```

Работа с записями

- Третий параметр `DmReleaseRecord` указывает на действительное изменение данных и устанавливает флажок изменения записи, который используется при синхронизации.
- Кроме полной перезаписи, можно изменять данные с некоторого смещения

```
VoidHand myRecord = DmGetRecord(gDB, recordNumber);  
StructType *s = MemHandleLock(myRecord);  
DmWrite(s, offsetof(StructType, field), &newValue,  
        sizeof(newValue));  
MemHandleUnlock(myRecord);  
DmReleaseRecord(gDB, recordNumber, true);
```

- Макрос `offsetof` вычисляет смещение поля в структуре.

Работа с записями

- Полное удаление записи осуществляется функцией

```
Err DmRemoveRecord (DmOpenRef dbP, UInt16 index)
```

- При этом могут возникнуть коллизии при синхронизации, если эта запись существовала раньше. Функция

```
Err DmRemoveRecord (DmOpenRef dbP, UInt16 index)
```

- удаляет данные, но оставляет заголовок записи с флагом удаления до ближайшей синхронизации. Сохранить данные до синхронизации можно с помощью архивирования

```
Err DmArchiveRecord (DmOpenRef dbP, UInt16 index)
```

Работа с записями

- Удаленные и архивированные записи следует перемещать в конец базы данных

```
if (isNew && !gSaveBackup)
DmRemoveRecord(gDB, recordNumber); // remove all traces
else {
if (gSaveBackup) //need to archive it on PC
DmArchiveRecord(gDB, recordNumber);
else
DmDeleteRecord(gDB, recordNumber); // leave ID and attrs
// Deleted records are stored at the end of the database
DmMoveRecord (gDB, recordNumber, DmNumRecords(gDB));
}
```


Данные приложения, категории

- Блок информации приложения -блок памяти, связанный с конкретной базой данных. В нем можно хранить, например, пользовательские настройки для данной базы, выбранный метод сортировки и/или список имен категорий.
- Обратите внимание на использование DmNewHandle для выделения памяти в той же куче, что содержит базу данных. Функция MemHandleNew выделяет память в динамической куче.

Пример кода:

```
UInt cardNo;
LocalID dbID;
LocalID appInfoID;
MyAppInfoType *appInfoP;
if (DmOpenDatabaseInfo(gDB, &dbID, NULL, NULL, &cardNo, NULL))
return dmErrInvalidParam;
h = DmNewHandle(gDB, sizeof(MyAppInfoType));
if (!h)
return dmErrMemError;
appInfoID = MemHandleToLocalID(h);
DmSetDatabaseInfo(cardNo, dbID, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, &appInfoID, NULL, NULL, NULL);
appInfoP = (MyAppInfoType *) MemHandleLock(h);
DmSet(appInfoP, 0, sizeof(MyAppInfoType), 0);
// initialize fields in appInfoP
// Unlock
MemPtrUnlock(appInfoP);
```

Категории

- Для использования менеджера категорий (Category Manager) необходимо первым полем данных приложения сделать структуру AppInfoType. Для инициализации этого поля используется

```
CategoryInitialize(&appInfoP->appInfo, LocalizedAppInfoStr);
```

- Вторым параметром - идентификатор ресурса "набор строк", который содержит исходные имена категорий.

Категории

- Функции менеджера категорий позволяют прочитать категории в список

```
void CategoryCreateList (DmOpenRef db, ListType *listP, UInt16  
    currentCategory, Boolean showAll, Boolean showUneditables,  
    UInt8 numUneditableCategories, UInt32 editingStrID, Boolean  
    resizeList)
```

- с последующим освобождением ресурсов в конце работы

```
void CategoryFreeList (DmOpenRef db, const ListType *listP,  
    Boolean showAll, UInt32 editingStrID)
```

Категории

- произвести обработку выбора в списке, включая редактирование списка категорий

```
Boolean CategorySelect (DmOpenRef db, const FormType *frm, UInt16  
    ctlID, UInt16 lstID, Boolean title, UInt16 *categoryP, char  
    *categoryName, UInt8 numUneditableCategories, UInt32  
    editingStrID)
```

- Например, обработка ctlSelectEvent для всплывающего списка (триггера)

```
// Process the category popup list.  
category = CurrentCategory;
```

```
frm = FrmGetActiveForm();  
categoryEdited = CategorySelect (ToDoDB, frm,  
    ListCategoryTrigger, ListCategoryList, true, &category,  
    CategoryName, 1, categoryDefaultEditCategoryString);
```

Категории

- Получить имя категории по ее индексу

```
void CategoryGetName (DmOpenRef db, UInt16 index, Char *name)
```

- Получить индекс следующей категории (индексы могут быть не последовательными)

```
UInt16 CategoryGetNext (DmOpenRef db, UInt16 index)
```

- Установить метку для элемента управления, соответствующую урезанному до нужного размера имени категории

```
void CategorySetTriggerLabel (ControlType *ctl, Char *name)
```

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

Разработка ПО для портативных компьютеров (КПК)

Palm OS (5)

Жерздев С.В.

Системные свойства

- В Palm OS предусмотрена возможность сохранения и чтения параметров системы и приложений. Каждое свойство характеризуется уникальным ID создателя, который обычно соответствует таковому для приложения, номером свойства и 32-разрядным значением. Константы для системных свойств определены в SystemMgr.h.

```
// See if we're on ROM version 2.0 or later.  
FtrGet(sysFtrCreator, sysFtrNumROMVersion, &romVersion);  
if (romVersion >= 0x02000000) {  
    ....  
}
```


Системные свойства

- Приложение может использовать свойства для хранения своих настроек.

```
Err FtrGet (UInt32 creator, UInt16 featureNum, UInt32 *valueP)
```

- Функция служит для чтения свойства и возвращает 0 в случае успеха.

- Для создания новых свойств и установки их значений служит функция

```
Err FtrSet (UInt32 creator, UInt16 featureNum, UInt32 newValue)
```

- Для удаления свойства используется

```
Err FtrUnregister (UInt32 creator, UInt16 featureNum)
```

Работа с датой и временем

- КПК на базе Palm OS имеет часы реального времени и программируемый таймер. Часы обеспечивают получение текущего времени с точностью 1 секунда и могут послать прерывание в заданный момент времени, в том числе «включить» устройство. Таймер используется в активном режиме и формирует сигналы каждую 0,01 секунды.
- Для чтения значений таймера (в тиках по 0,01 секунды с момента последнего «включения») используется функция

```
UInt32 TimGetTicks (void)
```

Часы реального времени

```
UInt32 TimGetSeconds (void)
```

- возвращает показания часов реального времени в секундах от 1 января 1904 года. Для преобразования этого значения в дату/время можно использовать функцию

```
void TimSecondsToDateTime (UInt32 seconds, DateTimePtr dateTimeP)
```

- Возможно и обратное преобразование

```
UInt32 TimDateTimeToSeconds (DateTimePtr dateTimeP)
```

Часы реального времени

- Структура для хранения даты/времени имеет следующий вид

```
typedef struct{
    Int16 second;
    Int16 minute;
    Int16 hour;
    Int16 day;
    Int16 month;
    Int16 year;
    Int16 weekDay ; //Days since Sunday (0 to 6)
}DateTimeType;
```

Преобразования даты

- Для преобразования даты и времени в текстовые строки используются функции

```
void DateToAscii (UInt8 months, UInt8 days, UInt16 years,  
    DateFormatType dateFormat, Char* pString)
```

- Здесь внешний вид строк определяется значениями

```
typedef enum {  
    dfMDYWithSlashes, // 12/31/95  
    dfDMYWithSlashes, // 31/12/95  
  
    ...  
  
    dfMDYLongWithComma, // Dec 31, 1995  
    dfDMYLong, // 31 Dec 1995  
} DateFormatType;
```

Преобразования времени

- Для преобразования даты и времени в текстовые строки используются функции

```
void TimeToAscii (UInt8 hours, UInt8 minutes, TimeFormatType  
    timeFormat, Char* pString)
```

```
typedef enum  
{  
    tfColon,  
    tfColonAMPM, // 1:00 pm  
    tfColon24h, // 13:00  
    tfDot,  
    tfDotAMPM, // 1.00 pm  
    tfDot24h, // 13.00  
    tfHoursAMPM, // 1 pm  
    tfHours24h, // 13  
    tfComma24h // 13,00  
} TimeFormatType;
```

Ввод даты/времени

- Для ввода (выбора пользователем) некоторой даты/времени/диапазона времени используются системные диалоговые окна. При отображении ЭТИХ ОКОН можно указать заголовок окна и исходное значение, в эти же переменные будет записано новое значение. Функции отображения возвращают true, если пользователь нажал ОК.

```
Boolean SelectDay (const SelectDayType selectDayBy, Int16 *month,  
                  Int16 *day, Int16 *year, const Char *title)
```

```
Boolean SelectOneTime (Int16 *hour, Int16 *minute, const Char  
                      *titleP)
```

```
Boolean SelectTime (TimeType * startTimeP, TimeType * endTimeP,  
                   Boolean untimed, const Char * titleP, Int16 startOfDay, Int16  
                   endOfDay, Int16 startOfDayDisplay)
```

Ввод даты/времени

- Значение

```
typedef enum
{
    selectDayByDay, // return d/m/y
    selectDayByWeek, // return d/m/y with d as same day of the week
    selectDayByMonth // return d/m/y with d as same day of the month
} SelectDayType;
```

- определяет тип диалогового окна (выбор даты, дня недели или дня месяца). Для задания времени используется структура

```
typedef struct {
    UInt8 hours;
    UInt8 minutes;
} TimeType;
```


Будильник

- Для выполнения периодических действий или отображения напоминаний Palm OS обеспечивает поддержку будильников (alarm) реального времени.
- Каждое приложение может установить один будильник, при срабатывании которого приложение будет вызвано с кодом запуска `sysAppLaunchCmdAlarmTriggered` и, затем с кодом `sysAppLaunchCmdDisplayAlarm`.

Будильник

- Для установки и сброса будильника используется функция

```
Err AlmSetAlarm (UInt16 cardNo, LocalID dbID, UInt32 ref, UInt32  
alarmSeconds, Boolean quiet)
```

- При этом указывается номер карты и идентификатор базы, содержащей приложение, параметр, который будет передан этому приложению, и время срабатывания в секундах от 1 января 1904 года (или 0 для сброса будильника). Последний параметр не используется.

Будильник

- Пример установки будильника

```
static void SetTimeOfNextAlarm (UInt32 alarmTime, UInt32 ref)
{
    UInt16 cardNo;
    LocalID dbID;
    DmSearchStateType searchInfo;

    DmGetNextDatabaseByTypeCreator (true, &searchInfo,
    sysFileTApplication, sysFileCDatebook, true, &cardNo, &dbID);

    AlmSetAlarm (cardNo, dbID, ref, alarmTime, true);
}
```

- Здесь sysFileTApplication определяет тип базы (приложение), а sysFileCDatebook задает ID создателя приложения.

Будильник

- Для обработки сработавшего будильника приложение должно обрабатывать код запуска `sysAppLaunchCmdAlarmTriggered`, с которым передается следующая информация

```
typedef struct SysAlarmTriggeredParamType {  
    UInt32 ref;  
    UInt32 alarmSeconds;  
    Boolean purgeAlarm;  
    UInt8 padding;  
} SysAlarmTriggeredParamType;
```

- По этому коду запуска приложение может установить новый будильник, проиграть звуковой сигнал, выполнить другую быструю обработку.

Будильник

- Для длительной обработки, в том числе отображения пользователю диалогового окна, используется код запуска `sysAppLaunchCmdDisplayAlarm`, сопровождаемый информацией

```
typedef struct SysDisplayAlarmParamType {  
    UInt32 ref;  
    UInt32 alarmSeconds;  
    Boolean soundAlarm;  
    UInt8 padding;  
} SysDisplayAlarmParamType;
```

Инфракрасный порт

- Для передачи данных через инфракрасный порт (beaming) используется менеджер обмена. В некоторых устройствах менеджер обмена может использоваться и для передачи другими средствами, например, через TCP/IP.
- Для реализации обмена следует
 1. Определить формат обмена (расширение файла или MIME-тип или оба средства) и формат передаваемых данных.
 2. Реализовать средства интерфейса пользователя для передачи данных (например, пункт меню Beam).
 3. Реализовать отправку данных приложением.
 4. Реализовать получение данных приложением.

Отправка данных

- Для организации обмена данными используются функции менеджера обмена, описанные в файле <ExtMgr.h>. Для установки соединения используется функция

```
Err ExgPut (ExgSocketPtr socketP)
```

- В качестве параметра передается указатель на заполненную структуру

```
typedef struct ExgSocketType {  
    UInt16 libraryRef;  
    UInt32 socketRef;  
    UInt32 target;  
    ...  
    Char *description;  
    Char *type;  
    Char *name;  
} ExgSocketType;
```

Отправка данных

- Все неиспользуемые поля должны быть заполнены нулями.
- Необходимо задать значения для полей:
 - description – текстовое описание передаваемого объекта для отображения пользователю
 - type, target, name – задают соответственно MIME-тип (не реализовано), ID создателя приложения или имя файла передаваемых данных. Можно задать одно из этих значений.
 - localmode – используется для отладки. Установите в 1 для передачи на локальное устройство или 0 для передачи через инфракрасный порт.

Отправка данных

- После установки соединения можно использовать функцию

```
UInt32 ExgSend (ExgSocketPtr socketP, const void * const bufP,  
                const UInt32 bufLen, Err * err)
```

- передав ей указатель на структуру соединения, указатель на буфер данных, длину передаваемых данных и указатель на переменную для кода ошибки. Функция блокирующая, возвращает объем реально переданных данных. Может вызываться многократно для передачи данных порциями. Может реализовать интерфейс пользователя для отмены передачи.

Отправка данных

- Пример использования

```
static Err BeamBytes(ExgSocketPtr s, void *buffer, ULong
    bytesToSend) {
    Err err = 0;
    while (!err && bytesToSend > 0) {
        ULong bytesSent = ExgSend(s, buffer, bytesToSend, &err);
        bytesToSend -= bytesSent;
        buffer = ((char *) buffer) + bytesSent;
    }
    return err;
}
```

- Завершается передача данных разрывом соединения:

```
Err ExgDisconnect(ExgSocketPtr socketP, Err error)
```

Прием данных

- Если используется идентификация данных по MIME-типу или по расширению, необходимо зарегистрировать приложение как приемник соответствующих данных.

```
Err ExgRegisterData (const UInt32 creatorID, const UInt16 id,  
    const Char * const dataTypesP)
```

- В качестве параметров указывается ID создателя приложения, регистрируемого как приемник, тип регистрации (по расширению exgRegExtensionID или по типу exgRegTypeID) и регистрируемое расширение (MIME-тип).

Прием данных

- При установке соединения соответствующее приложение получает код запуска `sysAppLaunchCmdExgAskUser`, обработка которого позволяет принять или отклонить соединение. Если этот код запуска не обрабатывается приложением, система запросит пользователя.

Прием данных

- Прием данных осуществляется в обработчике кода запуска `sysAppLaunchCmdExgReceiveData`. Для проверки того, что приложение уже запущено и база открыта, можно использовать флаги запуска

```
if (launchFlags & sysAppLaunchFlagSubCall) {  
    // приложение уже было запущено  
} else {  
    // открыть базу перед приемом данных  
}
```

- Для установки соединения на прием используется функция

```
Err ExgAccept (ExgSocketPtr socketP)
```

- Она заполняет структуру, используемую в последующих вызовах менеджера обмена.

Прием данных

- Примите данные в буфер путем вызова (возможно, многократного)

```
UInt32 ExgReceive (ExgSocketPtr socketP, void *bufP, const UInt32  
    bufLen, Err * err)
```

- Эта функция возвращает число действительно принятых байт. Возврат значения 0 означает конец данных или возникновение ошибки (можно определить по значению кода err).
- После завершения приема следует закрыть соединение с помощью ExgDisconnect, после чего приложение может отобразить принятые данные.

Связь по ТСР/ІР

- Palm OS предоставляет функции для реализации сетевых соединений в менеджере сетей. Эти функции обеспечивают низкоуровневое управление соединениями, приемом и передачей данных. Кроме того, в заголовочном файле `<unix/sys_socket.h>` описаны макросы для работы с сетью в стиле Berkeley sockets, например

```
#define bind(socket, localaddr, addrlen) \  
NetLibSocketBind(AppNetRefnum, socket, \  
(NetSocketAddrType*)localaddr, addrlen, AppNetTimeout, &errno)
```

Связь по ТСР/ІР

- Во многих ситуациях удобнее использовать базирующуюся на менеджере сетей библиотеку высокоуровневых соединений. Функции этой библиотеки описаны в заголовочном файле `<unix/sys_socket.h>`.
- Инициализация библиотеки производится
ВЫЗОВОМ

```
Err NetUInit (void);
```


Связь по ТСР/IP

- Установка соединения с сервером по протоколу ТСР осуществляется вызовом

```
NetSocketRef NetUTCPOpen (Char* hostName, Char* serviceName,  
    Int16 port)
```

- В качестве параметров передаются адрес сервера (DNS-имя или IP-адрес в виде строки), имя сервиса ("echo", "chargen", "ftp-data", "ftp", "telnet", "smtp", "time", "name", "pop3", "nntp", "imap2", ... или NULL для явного указания номера порта) и номер порта (если не указан сервис).
- Функция возвращает описатель соединения, используемый в функциях приема и передачи, или -1 в случае ошибки.

Связь по ТСР/ІР

- Для передачи и приема данных используются функции

```
Int32 NetUWriteN (NetSocketRef fd, UInt8* bufP, UInt32 numBytes)
```

```
Int32 NetUReadN (NetSocketRef fd, UInt8* bufP, UInt32 numBytes)
```

- Обе функции принимают описатель соединения, буфер и размер данных и возвращают количество реально переданных/принятых байт или -1 в случае ошибки.

- Для закрытия соединения можно использовать

```
int close(int socket)
```

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

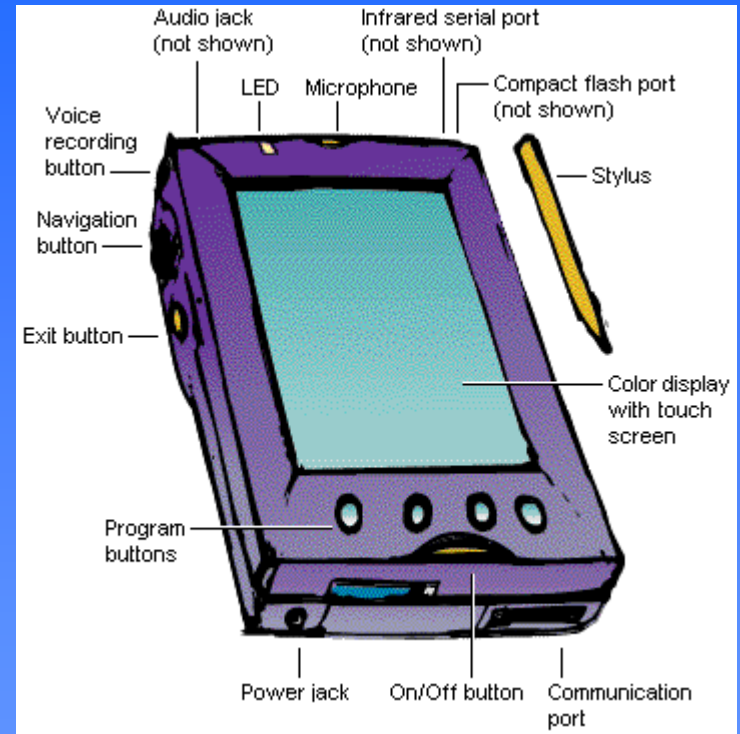
Разработка ПО для портативных компьютеров (КПК)

PocketPC

Жерздев С.В.

Платформа RocketPC

- Платформа RocketPC базируется на операционной системе Windows CE и охватывает разнообразные типы устройств, в том числе смартфоны и PDA.
- Устройства на платформе RocketPC производятся различными фирмами изготовителями и могут заметно отличаться по внешнему виду и комплектации (включая используемый процессор).



Типичное устройство PocketPC

- Сенсорный экран. Обеспечивает отображение интерфейса пользователя и средства ввода. Типичное разрешение 240*320, глубина цвета 16 бит.
- Стилус. Обеспечивает указание и ввод данных посредством сенсорного экрана и виртуальной клавиатуры.
- Элементы навигации. К ним относятся: кнопка питания, кнопка действия (аналог Enter), программные кнопки для запуска приложений, элемент прокрутки вверх/вниз, кнопка диктофона и др.

Типичное устройство PocketPC

- Аудио вход/выход.
- Средства оповещения. К ним могут относиться аудио сигналы, мигающий индикатор, вибросигнал и т.д.
- Батарея.
- Центральный процессор. Используется семейство процессоров ARM. Поддерживается компиляция для процессоров x86 для отладки в эмуляторе.
- Память. Устройство содержит не менее 24 MB ROM и 16 MB of RAM.
- Встроенный последовательный и инфракрасный порт.

Windows CE

- Microsoft® Windows® CE .NET – открытая, масштабируемая, 32-разрядная операционная система, разработанная для управления широким диапазоном устройств, включая промышленные контроллеры, коммуникационные устройства, торговые терминалы, Интернет приставки, интерактивные телевизоры. Windows CE .NET обеспечивает встроенную поддержку мультимедиа, Интернет, локальных сетей и мобильных коммуникаций.

Windows CE

- Операционная система Windows CE использует подмножество Microsoft Win32 API, что обеспечивает сходство разработки программ для PocketPC и настольных версий Windows, позволяет использовать Active Template Library (ATL) for Windows CE, Microsoft Foundation Classes (MFC) for Windows CE.

Windows CE

- Microsoft® Windows® CE .NET сочетает библиотеки Microsoft Win32® API, интерфейса пользователя (UI), и интерфейса графического устройства (GDI) в модуле Graphics, Windowing, and Events Subsystem (GWES). GWES обеспечивает взаимодействие пользователя, приложения и операционной системы. GWES поддерживает все окна, диалоги, элементы управления, меню и ресурсы, составляющие интерфейс пользователя.

Приложение PocketPC

- Приложение Pocket PC практически является приложением Windows, имеет цикл обработки событий, главное окно и процедуры окон. Однако есть и особенности.
- Во-первых, приложение Pocket PC должно обеспечить присутствие только одной своей копии в запущенном состоянии.
- Во-вторых, вместо использования панели команд (как другие приложения Windows CE), приложение Pocket PC должно использовать панель меню.

Приложение RocketPC

- Приложению Rocket PC не следует иметь кнопки Close, команд Exit или Close в своих меню. Пользователь не должен отслеживать состояние приложений, запущены они или нет, он должен пользоваться своим PDA.

Общая схема приложения RocketPC

- Функция WinMain, которая является точкой входа в приложение, помещает дескриптор экземпляра приложения в глобальную переменную. Таким образом, дескриптор становится доступен во всех функциях приложения.

```
int WINAPI WinMain(    HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPTSTR     lpCmdLine,
                      int         nCmdShow)
{
    MSG msg;
    HACCEL hAccelTable;

    g_hInst = hInstance; // Store instance handle
```

Общая схема приложения RocketPC

- Функция WinMain вызывает функцию InitApplication, к задачам которой обычно относится проверка на наличие запущенного экземпляра этого приложения, регистрация класса для основного окна приложения.

...

```
// Perform application initialization:  
if (!InitApplication ())  
{  
    return FALSE;  
}
```

...

Общая схема приложения RocketPC

- Функция WinMain вызывает функцию InitInstance для создания главного окна приложения посредством CreateWindow.

```
...  
// Perform application initialization:  
if (!InitInstance (hInstance, nCmdShow))  
{  
    return FALSE;  
}  
...
```

Общая схема приложения RocketPC

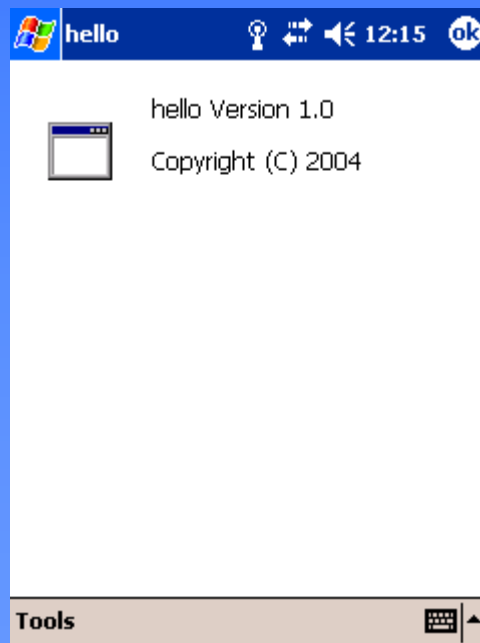
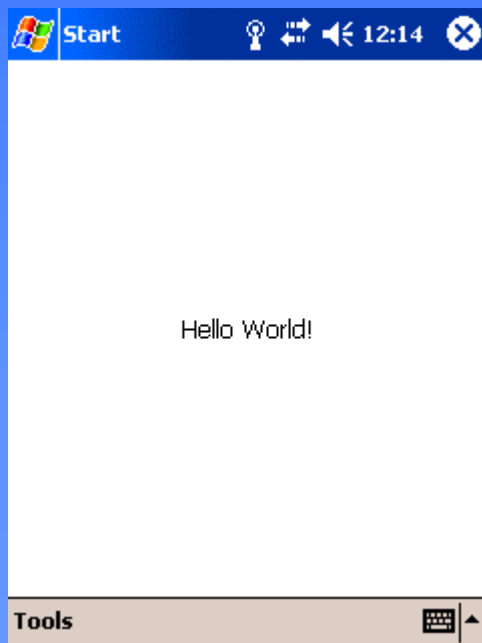
- Функция WinMain организует цикл обработки событий, вызывая GetMessage, TranslateMessage и DispatchMessage.

...

```
hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_HELLO);  
// Main message loop:  
while (GetMessage(&msg, NULL, 0, 0))  
{  
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))  
    {  
        TranslateMessage(&msg);  
        DispatchMessage(&msg);  
    }  
}  
return msg.wParam;  
}
```

Пример приложения

- Рассмотрим пример приложения для RocketPC. На рисунке показан его внешний вид в различных состояниях.



Файлы заголовков

```
// hello.cpp : Defines the entry point for the application.  
//  
#include "stdafx.h"  
#include "hello.h"  
#include <commctrl.h>  
#include <aygshell.h>  
#include <sipapi.h>
```

Константы и глоб. переменные

- Определяем константу для максимального размера загружаемых строк

```
#define MAX_LOADSTRING 100
```

- Глобальные переменные для дескрипторов экземпляра приложения и командной панели

```
HINSTANCE      g_hInst;           // The current instance  
HWND           g_hwndCB;          // The command bar handle
```

- Структура для состояния панели ввода

```
static SHACTIVATEINFO s_sai;
```

Декларация функций

```
// Forward declarations of functions included in this code
module:

ATOM          MyRegisterClass          (HINSTANCE, LPTSTR);
BOOL          InitInstance             (HINSTANCE, int);
LRESULT CALLBACK WndProc               (HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK About                  (HWND, UINT, WPARAM, LPARAM);
HWND          CreateRpCommandBar (HWND);
```

Основная функция приложения

```
int WINAPI WinMain(    HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPTSTR     lpCmdLine,
                      int         nCmdShow)
{
    MSG msg;
    HACCEL hAccelTable;
```

- Инициализация экземпляра (проверка на повторный запуск, регистрация класса окна и создание главного окна приложения)

```
// Perform application initialization:
if (!InitInstance (hInstance, nCmdShow))
{
    return FALSE;
}
```

Основная функция приложения

- Загрузка таблицы клавиш быстрого доступа

```
hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_HELLO);
```

- Цикл обработки событий

```
// Main message loop:  
while (GetMessage(&msg, NULL, 0, 0))  
{
```

- Преобразование клавиш быстрого доступа в события от пунктов меню

```
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))  
    {
```

Основная функция приложения

- Преобразование событий от виртуальных кнопок
в СИМВОЛЫ

```
TranslateMessage(&msg);
```

- Передача события в процедуру текущего окна

```
DispatchMessage(&msg);
```

```
}
```

```
}
```

```
return msg.wParam;
```

```
}
```

Регистрация класса окна

```
ATOM MyRegisterClass(HINSTANCE hInstance, LPTSTR szWindowClass)
{
    WNDCLASS      wc;

    wc.style          = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc    = (WNDPROC) WndProc;
    wc.cbClsExtra     = 0;
    wc.cbWndExtra     = 0;
    wc.hInstance      = hInstance;
    wc.hIcon          = LoadIcon(hInstance,
MAKEINTRESOURCE( IDI_HELLO ));
    wc.hCursor        = 0;
    wc.hbrBackground  = (HBRUSH) GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName   = 0;
    wc.lpszClassName  = szWindowClass;

    return RegisterClass(&wc);
}
```

Инициализация приложения

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd = NULL;
    TCHAR szTitle[MAX_LOADSTRING];           // The title bar text
    TCHAR szWindowClass[MAX_LOADSTRING];     // The window class name

    • Сохранить дескриптор экземпляра приложения
    g_hInst = hInstance;                     // Store instance handle in our
    global variable

    • Загрузить строки для класса окна и заголовка
    // Initialize global strings
    LoadString(hInstance, IDC_HELLO, szWindowClass,
    MAX_LOADSTRING);
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
```


Инициализация приложения

- Проверить на наличие запущенного экземпляра

```
//If it is already running, then focus on the window
hWnd = FindWindow(szWindowClass, szTitle);
if (hWnd)
{
    // set focus to foremost child window
    // The "| 0x01" is used to bring any owned windows to the
foreground and
    // activate them.
    SetForegroundWindow((HWND)((ULONG) hWnd | 0x00000001));
    return 0;
}
```

Инициализация приложения

- Зарегистрировать класс окна

```
MyRegisterClass(hInstance, szWindowClass);
```

- Создать главное окно

```
hWnd = CreateWindow(szWindowClass, szTitle, WS_VISIBLE,  
    CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,  
    CW_USEDEFAULT,  
    NULL, NULL, hInstance, NULL);  
if (!hWnd)  
{  
    return FALSE;  
}
```

Инициализация приложения

- Скорректировать высоту окна с учетом панели меню (оно создается в процедуре окна при вызове CreateWindow)

```
if (g_hwndCB)
{
    RECT rc;
    RECT rcMenuBar;

    GetWindowRect(hWnd, &rc);
    GetWindowRect(g_hwndCB, &rcMenuBar);
    rc.bottom -= (rcMenuBar.bottom - rcMenuBar.top);

    MoveWindow(hWnd, rc.left, rc.top, rc.right-rc.left,
rc.bottom-rc.top, FALSE);
}
```

Инициализация приложения

- Отобразить окно приложения

```
ShowWindow(hWnd, nCmdShow);
```

```
UpdateWindow(hWnd);
```

```
return TRUE;
```

```
}
```

Процедура главного окна

- Обрабатывает события, соответствующие этому окну.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
    LPARAM lParam)
{
    HDC hdc;
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    TCHAR szHello[MAX_LOADSTRING];

    switch (message)
    {
```

Процедура главного окна

- Обработка команд меню

```
case WM_COMMAND:
    wmId      = LOWORD(wParam);
    wmEvent   = HIWORD(wParam);
    // Parse the menu selections:
    switch (wmId)
    {
```

- Отображение диалогового окна из ресурса.
Указываем функцию-обработчик событий для диалога.

```
    case IDM_HELP_ABOUT:
        DialogBox(g_hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
(DLGPROC)About);
        break;
```

Процедура главного окна

- Закреть приложение, послав себе соответствующее событие.

```
case IDOK:
    SendMessage (hWnd, WM_CLOSE, 0, 0);
    break;
```

- Обработка команд по умолчанию

```
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
break;
```

Процедура главного окна

- Создание окна. Создаем панель меню и инициализируем структуру для состояния панели ввода

```
case WM_CREATE:
    g_hwndCB = CreateRpCommandBar(hWnd);
    // Initialize the shell activate info structure
    memset (&s_sai, 0, sizeof (s_sai));
    s_sai.cbSize = sizeof (s_sai);
    break;
```


Процедура главного окна

- Отрисовка окна

```
case WM_PAINT:
    RECT rt;
    hdc = BeginPaint(hWnd, &ps);
    GetClientRect(hWnd, &rt);
    LoadString(g_hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
    DrawText(hdc, szHello, _tcslen(szHello), &rt,
             DT_SINGLELINE | DT_VCENTER | DT_CENTER);
    EndPaint(hWnd, &ps);
    break;
```

Процедура главного окна

- Изменение параметров окна. Сохраняем состояние панели ввода

```
case WM_SETTINGCHANGE:  
    SHHandleWMSettingChange(hWnd, wParam, lParam, &s_sai);  
    break;
```

- Активация окна. Восстанавливаем параметры панели ввода

```
case WM_ACTIVATE:  
    // Notify shell of our activate message  
    SHHandleWMActivate(hWnd, wParam, lParam, &s_sai, FALSE);  
    break;
```

Процедура главного окна

- Уничтожение окна. Удаляем созданную окном панель и посылаем сообщение на закрытие приложения.

```
case WM_DESTROY:
    CommandBar_Destroy(g_hwndCB);
    PostQuitMessage(0);
    break;
```

- Обработка событий окна по умолчанию

```
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}
```

Создание панели меню из ресурса

```
HWND CreateRpCommandBar(HWND hwnd)
{
    SHMENUBARINFO mbi;

    memset(&mbi, 0, sizeof(SHMENUBARINFO));
    mbi.cbSize      = sizeof(SHMENUBARINFO);
    mbi.hwndParent  = hwnd;
    mbi.nToolBarId  = IDM_MENU;
    mbi.hInstRes    = g_hInst;
    mbi.nBmpId      = 0;
    mbi.cbBmpImages = 0;

    if (!SHCreateMenuBar(&mbi))
        return NULL;

    return mbi.hwndMB;
}
```

Обработчик события диалогового окна

```
// Mesage handler for the About box.  
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,  
    LPARAM lParam)  
{  
    SHINITDLGINFO shidi;  
  
    switch (message)  
    {
```

- Создание диалога. Создаем кнопку закрытия ОК

```
        case WM_INITDIALOG:  
            // Create a Done button and size it.  
            shidi.dwMask = SHIDIM_FLAGS;  
            shidi.dwFlags = SHIDIF_DONEBUTTON | SHIDIF_SIPDOWN  
            | SHIDIF_SIZEDLGFULLSCREEN;  
            shidi.hDlg = hDlg;  
            SHInitDialog(&shidi);  
            return TRUE;
```

Обработчик событий диалога

```
// Mesage handler for the About box.  
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,  
    LPARAM lParam)  
{  
    SHINITDLGINFO shidi;  
  
    switch (message)  
    {
```

- Создание диалога. Создаем кнопку закрытия ОК

```
        case WM_INITDIALOG:  
            // Create a Done button and size it.  
            shidi.dwMask = SHIDIM_FLAGS;  
            shidi.dwFlags = SHIDIF_DONEBUTTON | SHIDIF_SIPDOWN  
            | SHIDIF_SIZEDLGFULLSCREEN;  
            shidi.hDlg = hDlg;  
            SHInitDialog(&shidi);  
            return TRUE;
```

Обработчик событий диалога

- Обработка кнопки ОК. Закрываем диалог

```
        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK)
            {
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
            }
            break;
    }
    return FALSE;
}
```

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

Разработка ПО для портативных компьютеров (КПК)

PocketPC (2)

Жерздев С.В.

Элементы управления

- Элементы управления Windows CE, такие как кнопки, поля ввода или списки, представлены предопределенными классами окна, для которых в системе определены соответствующие процедуры окна.
- Поскольку элемент управления является окном, он создается вызовом `CreateWindow` или `CreateWindowEx` или автоматически при создании диалога из ресурса.

Элементы управления

- Существуют следующие predefined классы окна:
- **BUTTON** различные виды кнопок
- **EDIT** окно для ввода или отображения текста
- **LISTBOX** список строк
- **COMBOBOX** комбинация поля ввода и списка
- **STATIC** элемент для отображения статического текста или графики
- **SCROLLBAR** полоса прокрутки для окна

Пример

```
#define IDC_BUTTON 1028
HWND hwndButton;
hwndButton = CreateWindow(
    TEXT("BUTTON"),      //Control class name
    TEXT("Exit"),        //Button text
    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, //Button styles
    0,0,75,35,           //x, y, width, height
    hwndParent,          //Parent window
    (HMENU)IDC_BUTTON,   //Command identifier
    hInstance,           //Application HINSTANCE
    NULL);
```

- Обратите внимание на применение макроса TEXT для приведения строк к Unicode.
- Как и все дочерние окна, элементы управления автоматически удаляются при уничтожении родительского окна.

Сообщения

- Элементы управления оповещают родительское окно о событиях посредством сообщений WM_COMMAND, с параметрами, указывающими на событие HIWORD(wParam), ID элемента управления или пункта меню LOWORD(wParam) и дескриптор окна элемента управления (HWND)lParam.

Управление

- Управляются и настраиваются элементы управления также с помощью сообщений, но посылаемых уже самому элементу. Как правило, для этого используется блокирующая функция

```
LRESULT SendMessage (  
    HWND hWnd,  
    UINT Msg,  
    WPARAM wParam,  
    LPARAM lParam  
);
```

Управление

- Для посылки сообщений элементам диалогового окна используется аналогичная

```
LONG SendDlgItemMessage(  
    HWND hDlg,  
    int nIDDlgItem,  
    UINT Msg,  
    WPARAM wParam,  
    LPARAM lParam  
);
```

Кнопки

- Обычные кнопки создаются как окна класса `BUTTON` с указанием стиля `BS_PUSHBUTTON`. Кроме того, можно использовать стили `BS_BOTTOM`, `BS_CENTER`, `BS_LEFT`, `BS_RIGHT`, `BS_TOP`, `BS_VCENTER` для управления размещением текста и `BS_DEFPUSHBUTTON` для указания кнопки по умолчанию.
- При нажатии стилусом обычная кнопка формирует сообщение `WM_COMMAND` с кодом `BN_CLICKED`.

Флажок

- Флажок создается с использованием стиля BS_CHECKBOX или BS_3STATE (флажок с тремя состояниями). Можно использовать стили BS_AUTOCHECKBOX и BS_AUTO3STATE для флажков, автоматически меняющих свое состояние по клику на нем. При клике флажок формирует сообщение WM_COMMAND с кодом BN_CLICKED.

Флажок

- Для изменения состояния флажка ему можно послать сообщение `BM_SETCHECK`, при этом параметр `wParam` указывает новое состояние: `BST_UNCHECKED` – сброшен, `BST_CHECKED` – установлен, `BST_INDETERMINATE` – неактивен (только для `3STATE`). Событие `BM_GETCHECK` позволяет прочитать текущее состояние.

Пример

- Ручное переключение состояний в процедуре окна может быть реализовано так:

```
LRESULT BtnWnd (HWND hWnd, UINT wParam, WPARAM wParam, LPARAM lParam)
{
    INT i;

    // Since the Check Box button is not an auto check box, it
    // must be set manually.
    if ((LOWORD (wParam) == IDC_CHKBOX) &&
        (HIWORD (wParam) == BN_CLICKED)) {
        // Get the current state, complement, and set.
        i = SendDlgItemMessage (hWnd, IDC_CHKBOX, BM_GETCHECK, 0, 0);
        if (i == 0)
            SendDlgItemMessage (hWnd, IDC_CHKBOX, BM_SETCHECK, 1, 0);
        else
            SendDlgItemMessage (hWnd, IDC_CHKBOX, BM_SETCHECK, 0, 0);
        return 0;
    }
    else
        return DefWindowProc (hWnd, wParam, wParam, lParam);
}
```

Переключатели

- Переключатели создаются и управляются аналогично, но используется стиль BS_RADIOBUTTON или BS_AUTORADIOBUTTON.

Поле ввода

- Поле ввода обеспечивает пользователю ввод текстовых данных с поддержкой навигации, операций удаления, копирования и вставки. Поле ввода отображает одну или, при использовании стиля `ES_MULTILINE` совместно с `ES_WANTRETURN`, несколько строк текста.

Поле ввода

- Для обеспечения автоматической прокрутки можно использовать стили ES_AUTOHSCROLL и ES_AUTOVSCROLL. Выравнивание текста можно задать стилями ES_RIGHT, ES_LEFT, ES_CENTER. Тип вводимых данных определяется стилями ES_PASSWORD, ES_NUMBER, ES_LOWERCASE, ES_UPPERCASE. Защитить текст от изменения можно стилем ES_READONLY.

Поле ввода

- Изменить текст в поле ввода можно сообщением WM_SETTEXT (wParam = 0; lParam = (LPARAM)(LPCTSTR) lpsz;). Для чтения текста в буфер используется сообщение WM_GETTEXT (wParam = (WPARAM) cchTextMax; lParam = (LPARAM) lpszText;).

Списки

- Списки используются для отображения набора строк и обеспечения возможности выбора пользователем одной или нескольких (стиль `LBS_MULTIPLESEL`) из них, возможность выделения можно отключить (`LBS_NOSEL`) и обрабатывать только сообщения от кликов на элементах (`LBS_NOTIFY`). Списки могут обеспечить сортировку элементов (стиль `LBS_SORT`) и прокрутку. Есть возможность организации списков с несколькими колонками (`LBS_MULTICOLUMN`).

Списки

- Можно динамически добавлять элементы в список, посылая ему сообщение LB_ADDSTRING или LB_INSERTSTRING (вставка в позицию wParam), указатель на строку заносится в lParam. Удаление элементов производится сообщением LB_DELETESTRING (wParam = (LPARAM) index;).
- Поиск строк в списке можно производить как по префиксу (LB_FINDSTRING), так и по целой строке (LB_FINDSTRINGEXACT). При этом wParam указывает элемент перед началом поиска (-1 – поиск с начала списка), lParam – указатель на строку.

Списки

- Для определения выделенного элемента можно использовать сообщение LB_GETCURSEL или пары сообщений (для множественного выделения) LB_GETSELCOUNT, LB_GETSELITEMS (wParam = (WPARAM) cItems; lParam = (LPARAM)(LPINT) lpnItems;). Проверить состояние конкретного элемента можно сообщением LB_GETSEL (wParam = (WPARAM) index;).

Списки

- Для установки выделения используются сообщения LB_SETCURSEL (wParam = (WPARAM) index;) и LB_SETSEL (wParam = (WPARAM)(BOOL) fSelect; lParam = (LPARAM)(UINT) index;).

Комбинированный список

- Вид и поведение этого управляющего элемента определяется следующими стилями: CBS_SORT, CBS_UPPERCASE, CBS_LOWERCASE, CBS_DROPDOWN (список скрыт и разворачивается по требованию пользователя), CBS_DROPDOWNLIST (возможен только выбор элементов в списке).
- Поскольку этот элемент совмещает свойства поля ввода и списка, его сообщения дублируют сообщения для этих элементов. К основным сообщениям можно отнести CB_ADDSTRING, CB_DELETESTRING, CB_FINDSTRING, CB_FINDSTRINGEXACT, CB_GETCURSEL и др.

Статический элемент

- Статический элемент обеспечивает отображение текста, графических изображений и иконок. Он не предусматривает взаимодействия с пользователем, но может сообщать о кликах в области элемента (стиль `SS_NOTIFY`). Для этого используется сообщение `STN_CLICKED` (`idStatic = (int) LOWORD(wParam); hwndStatic = (HWND) LOWORD(lParam);`).
- С помощью стилей можно указать параметры отображения текста – `SS_LEFT`, `SS_CENTER`, `SS_RIGHT`, `SS_LEFTNOWORDWRAP`.

Статический элемент

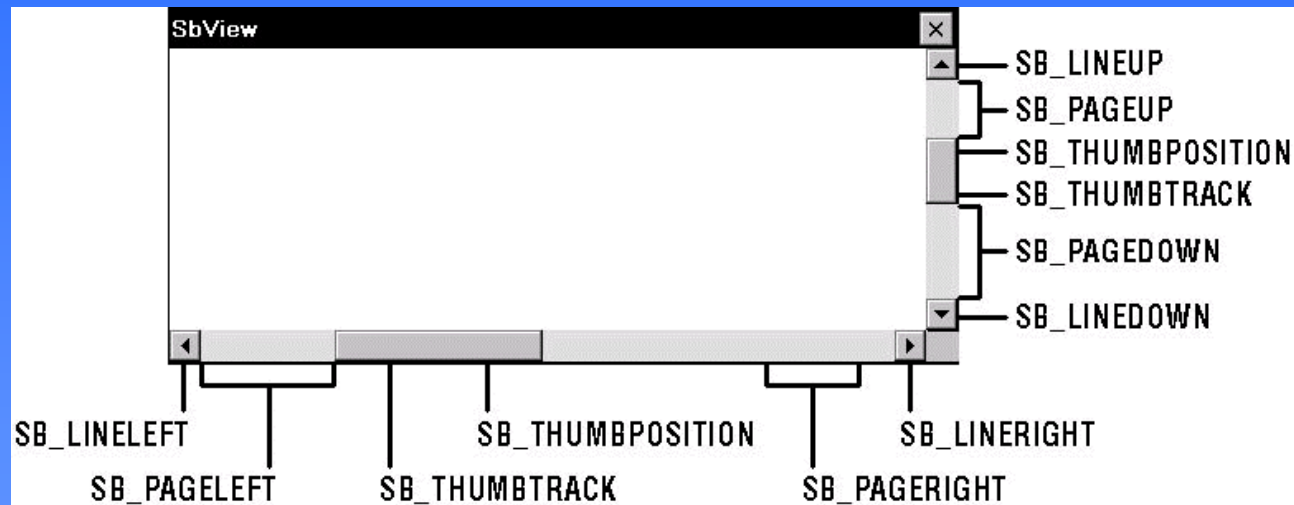
- Для отображения растрового изображения или иконки из ресурса применяются стили SS_BITMAP и SS_ICON соответственно. В этом случае текст окна задает имя изображения в ресурсе (не имя файла!). Совместно с этими стилями для центрирования изображения в элементе может применяться стиль SS_CENTERIMAGE.

Полоса прокрутки

- Направление полосы прокрутки задается стилями SBS_VERT и SBS_HORZ. Для привязки к границе родительского окна можно использовать стили SBS_LEFTALIGN и SBS_TOPALIGN соответственно.
- Родительское окно для слежения за состоянием полосы прокрутки должно обрабатывать сообщения WM_VSCROLL и WM_HSCROLL. Эти сообщения сопровождаются параметрами:
 - `nScrollCode = (int)LOWORD(wParam);`
 - `nPos = (short int)HIWORD(wParam);`
 - `hwndScrollBar = (HWND) lParam;`

Полоса прокрутки

- Значения nScrollCode определяют тип события, некоторые варианты представлены на рисунке.



Полоса прокрутки

- Параметр `nPos` определяет текущее положение для сообщений `SB_THUMBPOSITION` or `SB_THUMBTRACK`. В других случаях это значение не используется.
- значение `nPos` 16-разрядное. Для чтения 32-разрядного значения можно использовать, например,

```
BOOL GetScrollInfo(  
    HWND hwnd, // дескриптор элемента или окна  
    int fnBar, // SB_CTL (элемент управления), SB_HORZ, SB_VERT  
    LPSCROLLINFO lpsi // структура для значений  
);
```


Изменение положения

```
int SetScrollPos(  
    HWND hWnd, // дескриптор элемента или окна  
    int nBar, // SB_CTL (элемент управления), SB_HORZ, SB_VERT  
    int nPos, // новое положение  
    BOOL bRedraw // требуется перерисовка  
);
```

- **Параметры полосы прокрутки можно установить с ПОМОЩЬЮ**

```
int SetScrollInfo(  
    HWND hwnd,  
    int fnBar,  
    LPSCROLLINFO lpsi,  
    BOOL fRedraw  
);
```

Структура параметров

```
typedef struct tagSCROLLINFO {  
    UINT cbSize; // размер структуры  
    UINT fMask; // устанавливаемые или получаемые параметры  
    int nMin;  
    int nMax;  
    UINT nPage;  
    int nPos;  
    int nTrackPos; // текущее положение при протяжке, только для  
    чтения  
} SCROLLINFO;
```

- Для установки и чтения 16-разрядных значений параметров можно использовать и сообщения SBM_GETPOS, SBM_GETRANGE, SBM_SETPOS, SBM_SETRANGE.

Пример обработки сообщения

```
case WM_VSCROLL:
{
    int nScrollCode = (int)LOWORD(wParam);
    int nPos = (short int)HIWORD(wParam);

    SCROLLINFO si = {sizeof(SCROLLINFO),
                     SIF_PAGE|SIF_POS|SIF_RANGE|SIF_TRACKPOS, 0, 0, 0, 0,
                     0};
    GetScrollInfo (hWnd, SB_VERT, &si);

    int nNewPos = si.nPos;

    switch (nScrollCode)
    {
        // Include code that checks for other values of nScrollCode.
        // ...
        case SB_THUMBPOSITION:
            nNewPos = nPos + si.nMin; // Adding si.nMin is the workaround.
            break;
    }

    si.fMask = SIF_POS;
    si.nPos = nNewPos;
    SetScrollInfo (hWnd, SB_VERT, &si, TRUE);
}

return TRUE;
```

Диалоговые окна

- Для создания диалогового окна по описанию в ресурсах используется макрос DialogBox – надстройка над функцией CreateWindowEx.

```
int DialogBox(  
    HINSTANCE hInstance,  
    LPCTSTR lpTemplate,  
    HWND hWndParent,  
    DLGPROC lpDialogFunc  
);
```

- Параметры функции определяют дескриптор приложения, имя шаблона в ресурсе, родительское окно и функцию обработки сообщений диалога.
- Возвращаемое значение – параметр, переданный в функцию EndDialog в функции обработки событий.

Обработка сообщений

- Функция обработки сообщений диалога является частным случаем процедуры окна, но имеет некоторые особенности.
- Во-первых, если сообщение не обработано, она не должна вызывать DefWindowProc. Вместо этого, функция возвращает FALSE. В этом случае, функция обработки сообщений по умолчанию вызывается автоматически.
- Во-вторых, необходимо обрабатывать дополнительное сообщение WM_INITDIALOG. При этом производится вся необходимая инициализация диалогового окна перед отображением.

Обработка сообщений

- Использование сообщения WM_CREATE для инициализации управляющих элементов не сработает, т.к. на этот момент управляющие элементы еще не созданы.
- Большинству диалоговых окон нет необходимости обрабатывать сообщение WM_PAINT, т.к. оно не содержит ничего, кроме элементов управления.

Обработка сообщений

- Основная задача – обработка сообщений от элементов управления. Кроме того, необходимо обрабатывать сообщение WM_COMMAND со специальными ID элемента. Это IDOK и IDCANCEL. IDOK привязывается кнопке ОК на заголовке диалогового окна, а IDCANCEL соответствует кнопке Close.

Обработка сообщений

- В ответ на сообщения о закрытии, функция обработки должна произвести необходимые действия (например, сохранить новые значения) и вызвать функцию

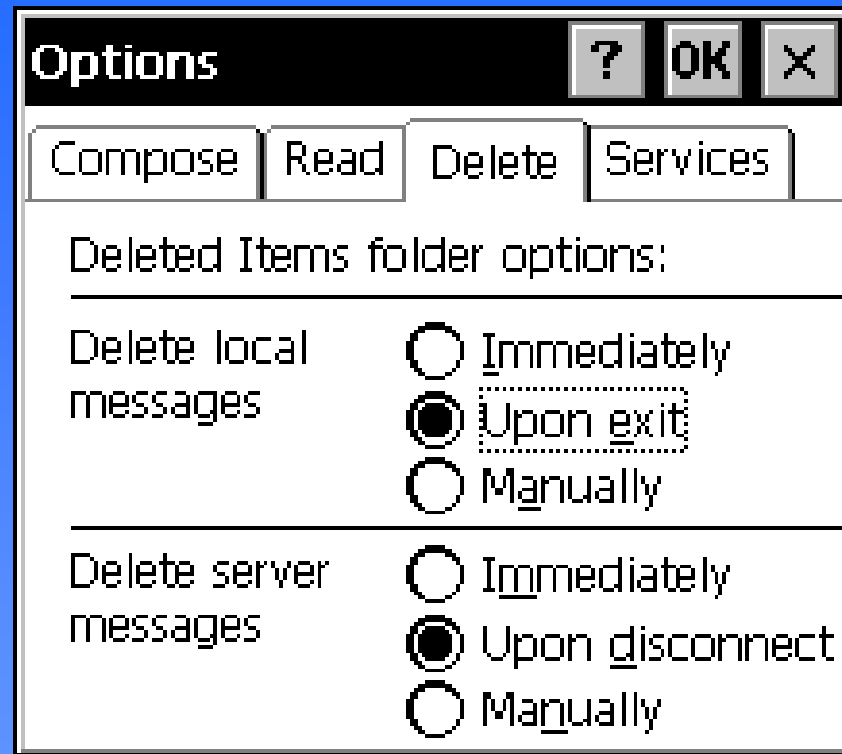
```
BOOL EndDialog (HWND hDlg, int nResult);
```

- которая закрывает диалоговое окно и возвращает управление в точку создания диалогового окна. Первый параметр – дескриптор закрываемого диалога, второй – значение, которое будет возвращено функцией создания диалогового окна.

Поля свойств

- С точки зрения пользователя, поле свойств – это диалоговое окно с одной или несколькими закладками, которые позволяют переключаться между «страницами» диалогового окна. С точки зрения программы, поле свойств – набор диалоговых окон, из которых видимым является только одно.
- Каждая страница поля свойств определяется шаблоном диалогового окна в ресурсах или созданным динамически. Каждая страница имеет собственную процедуру обработки сообщений. Кнопки ОК и Cancel располагаются в заголовке, кнопка Apply не предусмотрена.

Пример поля свойств



Создание поля свойств

- Для создания поля свойств используется функция
`int PropertySheet (LPCPROPSHEETHEADER lppsph);`
- В качестве параметра передается указатель на структуру, описывающую поле свойств в целом

Структура поля свойств

```
typedef struct _PROPSHEETHEADER {
    DWORD dwSize; // размер структуры
    DWORD dwFlags; // вид и поведение поля
                    // PSH_PROPSHEETPAGE для ссылок на структуры страниц
    HWND hwndOwner; // родительское окно
    HINSTANCE hInstance; // дескриптор приложения
    union { // не используется, должно быть 0
        HICON hIcon;
        LPCWSTR pszIcon;
    };
    LPCWSTR pszCaption; // заголовок
    UINT nPages; // число страниц
    union { // стартовая страница (номер или название)
        UINT nStartPage;
        LPCWSTR pStartPage;
    };
    union { // массив указателей на структуры страниц или сами страницы
        LPCPROPSHEETPAGE ppsp;
        HPROPSHEETPAGE FAR *phpage;
    };
    PFNPROPSHEETCALLBACK pfnCallback; // функция настройки поля,
                                        // по умолчанию не используется
} PROPSHEETHEADER;
```

Создание страниц

- Для создания страниц по структуре описания страницы используется

```
HPROPSHEETPAGE CreatePropertySheetPage (LPCPROPSHEETPAGE lppsp);
```

Структура описания страницы

```
typedef struct _PROPSHEETPAGE {
    DWORD dwSize; // размер структуры
    DWORD dwFlags;
    HINSTANCE hInstance; // дескриптор приложения, содержащего ресурсы
    union { // шаблон страницы, по умолчанию используется имя ресурса
        LPCSTR pszTemplate;
        LPCDLGTEMPLATE pResource;
    };
    union { // не используется, должно быть 0
        HICON hIcon;
        LPCSTR pszIcon;
    };
    LPCSTR pszTitle; // заголовок закладки
    DLGPROC pfnDlgProc; // функция обработки сообщений
    LPARAM lParam; // дополнительный параметр сообщений
    LPFNPSPCALLBACK pfnCallback; // функция настройки страницы,
                                // по умолчанию не используется
    UINT FAR * pcRefParent; // по умолчанию не используется
} PROPSHEETPAGE;
```

Пример создания поля свойств

```
PROPSHEETHEADER psh;
PROPSHEETPAGE psp[3];
INT i;

// Initialize page structures with generic information.
memset (&psp, 0, sizeof (psp));          // Zero out all unused values.
for (i = 0; i < dim(psp); i++) {
    psp[i].dwSize = sizeof (PROPSHEETPAGE);
    psp[i].dwFlags = PSP_DEFAULT;         // No special processing needed
    psp[i].hInstance = hInst;             // Instance handle where the
}                                          // dialog templates are located
// Now do the page-specific stuff.
psp[0].pszTemplate = TEXT ("Page1"); // Name of dialog resource for page 1
psp[0].pfnDlgProc = Page1DlgProc;      // Pointer to dialog proc for page 1

psp[1].pszTemplate = TEXT ("Page2"); // Name of dialog resource for page 2
psp[1].pfnDlgProc = Page2DlgProc;      // Pointer to dialog proc for page 2

psp[2].pszTemplate = TEXT ("Page3"); // Name of dialog resource for page 3
psp[2].pfnDlgProc = Page3DlgProc;      // Pointer to dialog proc for page 3
```

Пример создания поля свойств

```
// Init property sheet header structure.
psh.dwSize = sizeof (PROPSHEETHEADER);
psh.dwFlags = PSH_PROPSHEETPAGE;      // We are using templates, not handles.
psh.hwndParent = hWnd;                 // Handle of the owner window
psh.hInstance = hInst;                 // Instance handle of the application
psh.pszCaption = TEXT ("Property sheet title");
psh.nPages = dim(psp);                  // Number of pages
psh.nStartPage = 0;                     // Index of page to be shown first
psh.ppsp = psp;                         // Pointer to page structures
psh.pfnCallback = 0;                    // We don't need a callback procedure.

// Create property sheet.  This returns when the user dismisses the sheet
// by tapping OK or the Close button.
i = PropertySheet (&psh);
```


Процедура обработки сообщений

- lParam содержит указатель на структуру описания страницы.
- Страница не должна обрабатывать сообщений от IDOK и IDCANCEL и, соответственно, вызывать EndDialog. Вместо этого следует обрабатывать сообщение WM_NOTIFY. С этим событием параметр lParam содержит указатель на структуру

```
typedef struct tagNMHDR {  
    HWND hwndFrom; // окно страницы  
    UINT idFrom;    // ID элемента управления  
    UINT code;      // код события, может указывать на  
                   // существование  
                   // дополнительных полей в структуре  
} NMHDR;
```

Переключение страниц

- При переключении страниц текущая страница получает сообщение WM_NOTIFY с кодом PSN_KILLACTIVE. Процедура страницы должна проверить введенные данные на корректность. В случае, если данные корректны, надо установить поле возвращаемого значения в структуре окна в PSNRET_NOERROR и вернуть значение TRUE. Для установки поля возвращаемого значения используется подобная конструкция:

```
SetWindowLong (hwndPage, DWL_MSGRESULT, PSNRET_NOERROR);
```

Переключение страниц

- Для сохранения фокуса надо передать в этом вызове PSNRET_INVALID_NOCHANGEPAGE. Страница, получающая управление, получит сообщение WM_NOTIFY с кодом PSN_SETACTIVE.

Заккрытие поля свойств

- Когда пользователь закрывает поле свойств, нажимая на кнопку ОК, текущая страница также получит сообщение WM_NOTIFY с кодом PSN_KILLACTIVE, после чего все страницы получат сообщение WM_NOTIFY с кодом PSN_APPLY. Каждая страница должна при этом сохранить введенные пользователем данные.
- При нажатии на кнопку Close, текущая страница получит уведомление PSN_QUERYCANCEL. Процедура должна вернуть TRUE для предотвращения закрытия или FALSE для его разрешения. После этого все страницы получат уведомление PSN_RESET.

Системные диалоговые окна

- В системе предусмотрено несколько стандартных диалоговых окон, например, для выбора цвета, выбора имени файла для открытия или создания, печати.
- Как правило, для их вызова предусмотрены простые функции, например для открытия файла на чтение

```
BOOL GetOpenFileName(  
    LPOpenFileName lpofn  
);
```

- И на запись

```
BOOL GetSaveFileName(  
    LPOpenFileName lpofn  
);
```

Системные диалоговые окна

- в обоих случаях используется структура

```
typedef struct tagOFN { /* ofn */
    DWORD lStructSize;
    HWND hwndOwner;
    HINSTANCE hInstance;
    LPCSTR lpstrFilter;
    LPSTR lpstrCustomFilter;
    DWORD nMaxCustFilter;
    DWORD nFilterIndex;
    LPSTR lpstrFile;
    DWORD nMaxFile;
    LPSTR lpstrFileTitle;
    DWORD nMaxFileTitle;
    LPSTR lpstrInitialDir;
    LPCSTR lpstrTitle;
    DWORD Flags;
    WORD nFileOffset;
    WORD nFileExtension;
    LPCSTR lpstrDefExt;
    DWORD lCustData;
    LPOFNHOOKPROC lpfnHook;
    LPCSTR lpTemplateName;
} OPENFILENAME;
```

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

Разработка ПО для портативных компьютеров (КПК)

PocketPC (3)

Жерздев С.В.

Архитектура памяти

- Память, используемая Windows CE, делится на ROM и RAM.
- Типичный размер ROM – от 4 до 32 Мб. Эта память содержит всю операционную систему и приложения, поставляемые с платформой. Некоторые исполняемые модули могут выполняться непосредственно в ROM, без копирования их в страницы RAM.

Архитектура памяти

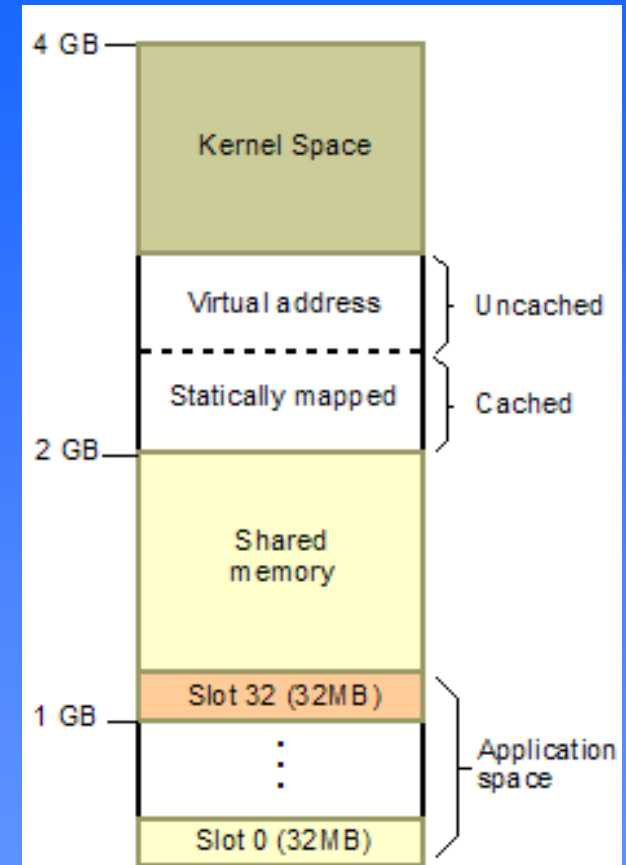
- RAM на устройствах Windows CE делится на две области: хранилище объектов (object store) и программную память (program memory).
Хранилище объектов похоже на виртуальный диск. Данные в этой области сохраняются при «выключении» и мягкой перезагрузке устройства. Программная память используется как RAM в ПК – она содержит кучи и стеки приложений, которые запущены в данный момент.

Архитектура памяти

- Программы, размещенные в хранилище объектов RAM или на дополнительных картах памяти, не могут исполняться по месту и предварительно копируются в программную область RAM и лишь после этого исполняются.
- Граница между хранилищем объектов и программной памятью может быть передвинута пользователем в ту или иную сторону с использованием System Control Panel или самой системой в случае необходимости.

Архитектура памяти

- При инициализации Windows CE создается единое виртуальное адресное пространство объемом 4 Гб. Оно делится на 33 слота по 32 Мб. Все процессы исполняются в одном адресном пространстве. При запуске процесса ОС выбирает свободный слот в адресном пространстве. Нулевой слот зарезервирован для текущего исполняемого процесса.



Архитектура памяти

- Ядро Windows CE использует для размещения и управления памятью систему виртуальной памяти на основе страниц. Система виртуальной памяти выделяет непрерывные области (regions) по 64 Кб, состоящие из 1024- или 4096-байтовых страниц. Если приложению требуется менее 64 Кб, можно использовать локальную кучу, предоставляемую системой для каждого приложения или создать отдельную кучу. Кроме того, ядро обеспечивает отдельный стек для каждого процесса и потока.

Архитектура памяти

- Приложение может выделить память за пределами своего 32 Мб слота, используя отображаемые в память файлы или непосредственно обращаясь к системе виртуальной памяти.
- Для доступа к виртуальной памяти, локальной и отдельным кучам, стеку, предусмотрены функции ядра. Для разделения данных между приложениями можно применять отображаемые в память объекты, например, файлы.

Виртуальная память

- Виртуальная память – основной вид памяти, который используется системой для организации всех остальных, включая кучи и стеки. API виртуальной памяти включает функции для выделения, фиксации в физической памяти и освобождения страниц памяти.
- Для выделения и резервирования памяти используется функция

```
LPVOID VirtualAlloc (LPVOID lpAddress, DWORD dwSize,  
                    DWORD flAllocationType,  
                    DWORD flProtect);
```

Виртуальная память

- Первый параметр – виртуальный адрес области памяти для выделения. Этот параметр используется для выделения блока в предварительно зарезервированной области. Если этот параметр равен NULL, система определяет подходящий адрес самостоятельно (с выравниванием на границу 64 Кб).
- Второй параметр – размер области для выделения или резервирования в байтах. Система округлит размер выделяемой области до ближайшей границы страницы.

Виртуальная память

- Параметр `flAllocationType` определяет тип операции и может быть комбинацией следующих флагов:
 - `MEM_COMMIT` – выделить память для использования программой,
 - `MEM_RESERVE` – зарезервировать пространство адресов для последующего выделения с использованием флага `MEM_COMMIT`,
 - `MEM_AUTO_COMMIT` – зарезервировать область адресов с автоматическим выделением страниц при первом обращении (только Windows CE),
 - `MEM_TOP_DOWN` – пытаться выделить память в верхних адресах.

Виртуальная память

- VirtualAlloc может вызываться многократно для одних и тех же областей, что позволяет производить выделение памяти в зарезервированной области без предварительной проверки.

Виртуальная память

- Параметр flProtect определяет флаги доступа к выделяемой области памяти:

- PAGE_READONLY
- PAGE_READWRITE
- PAGE_EXECUTE
- PAGE_EXECUTE_READ
- PAGE_EXECUTE_READWRITE
- PAGE_GUARD
- PAGE_NOACCESS
- PAGE_NOCACHE

- Изменение прав доступа к области

```
BOOL VirtualProtect (LPVOID lpAddress, DWORD dwSize,  
                    DWORD flNewProtect, PDWORD lpflOldProtect);
```

Виртуальная память

- Области виртуальной памяти резервируются по 64 Кб, выделение памяти в зарезервированной области происходит постранично.
- Исходя из этого, в 32 Мб адресном пространстве процесса может быть зарезервировано не более 511 областей виртуальной памяти (на практике около 475, т.к. часть областей используется для кода, кучи, стеков и т.п.). Например:

```
#define PAGE_SIZE 1024    // Assume we're on a 1-KB page machine
for (i = 0; i < 512; i++)
    pMem[i] = VirtualAlloc (NULL, PAGE_SIZE, MEM_RESERVE |
        MEM_COMMIT, PAGE_READWRITE);
```

Виртуальная память

- Правильнее было бы использовать такой способ:

```
#define PAGE_SIZE 1024    // Assume we're on a 1-KB page machine.

// Reserve a region first.
pMemBase = VirtualAlloc (NULL, PAGE_SIZE * 512, MEM_RESERVE,
                        PAGE_NOACCESS);

for (i = 0; i < 512; i++)
    pMem[i] = VirtualAlloc (pMemBase + (i*PAGE_SIZE), PAGE_SIZE,
                        MEM_COMMIT, PAGE_READWRITE);
```

Виртуальная память

- Для освобождения областей виртуальной памяти используется функция

```
BOOL VirtualFree (LPVOID lpAddress, DWORD dwSize, DWORD dwFreeType);
```

- Флаг MEM_DECOMMIT указывает на необходимость освобождения физических страниц, но область адресного пространства остается зарезервированной. MEM_RELEASE освобождает страницы и область адресного пространства (значение dwSize должно быть 0).
- Все страницы, освобождаемые VirtualFree, должны быть в одном состоянии (зарезервированные или выделенные).

Кучи

- Очевидно, что постраничное выделение памяти неудобно для большинства приложений. Система предоставляет возможность выделения небольших областей памяти с использованием механизма куч (heaps). Использование куч также ограждает приложение от особенностей конкретных архитектур (размер страницы).
- Кучи – области виртуальной памяти, выделяемые и управляемые системой. Система предоставляет функции для выделения и освобождения областей в пределах кучи. В случае необходимости система производит выделение или освобождение страниц кучи.

Кучи

- В отличие от настольных версий Windows, Windows CE поддерживает только фиксированные блоки в куче. Это облегчает использование блоков, но приводит к постепенной фрагментации кучи.
- Каждое приложение имеет кучу по умолчанию или так называемую локальную кучу. Эта куча создается при запуске приложения. Кроме того, приложение может создать произвольное число отдельных куч. Управление отдельными кучами во многом совпадает с управлением локальной.

Локальная куча

- Для размещения блока в локальной куче используется функция

```
HLOCAL LocalAlloc (UINT uFlags, UINT uBytes);
```

- Функция возвращает дескриптор блока памяти. Тем не менее, поскольку используются только фиксированные блоки, возвращаемое значение можно непосредственно использовать как указатель на выделенный блок памяти.

Локальная куча

- Значения параметра `uFlags` описывают характеристики выделяемого блока и могут быть следующими:
 - `LMEM_FIXED` Выделение фиксированного блока. Поскольку все блоки фиксированные, этот флаг избыточен.
 - `LMEM_ZEROINIT` Инициализировать выделяемый блок нулями.
 - `LPTR` Комбинация предыдущих флагов.
- Параметр `uBytes` определяет размер выделяемого блока в байтах. Фактический размер блока будет округлен до границы 4 байт.

Локальная куча

- Освобождение блока в локальной куче производится вызовом

```
HLOCAL LocalFree (HLOCAL hMem);
```

- Эта функция возвращает NULL в случае успеха или исходное значение дескриптора в случае неудачи.

Локальная куча

- Для изменения размера выделенного блока используется вызов

```
HLOCAL LocalReAlloc (HLOCAL hMem, UINT uBytes, UINT uFlag);
```

- В качестве параметров указываются текущий дескриптор блока, желаемый размер и один из флагов. Допустимые значения флагов — LMEM_ZEROINIT для заполнения нулями добавляемой части и LMEM_MOVEABLE если допускается перемещение блока в куче. В последнем случае возвращаемый дескриптор может отличаться от исходного.

Локальная куча

- Определить размер блока можно вызовом

```
UINT LocalSize (HLOCAL hMem);
```

Отдельные кучи

- Для предотвращения фрагментации локальной кучи можно использовать набор отдельных куч для различных задач. Память для отдельных куч выделяется по мере необходимости и освобождается по возможности.

Отдельные кучи

- Для создания отдельной кучи используется функция

```
HANDLE HeapCreate (DWORD flOptions, DWORD dwInitialSize, DWORD dwMaximumSize);
```

- Первый параметр может быть NULL или HEAP_NO_SERIALIZE (игнорируется, обращения из разных потоков всегда сериализуются).

Отдельные кучи

- Параметры `dwInitialSize` и `dwMaximumSize` определяют начальный и ожидаемый максимальный размер кучи. Не используются в ранних версиях Windows CE, но надо указывать реальные значения с учетом следующих версий.
- Выделение, освобождение и управление размером блоков происходит практически идентично локальной куче.

Отдельные кучи

```
LPVOID HeapAlloc (HANDLE hHeap, DWORD dwFlags, DWORD dwBytes);
```

- Возвращаемое значение является просто указателем, т.к. даже в настольных версиях Windows отдельные кучи поддерживают только фиксированные блоки. Флаг может принимать значения `HEAP_NO_SERIALIZE` и `HEAP_ZERO_MEMORY`.

```
BOOL HeapFree (HANDLE hHeap, DWORD dwFlags, LPVOID lpMem);
```

- Единственное возможное значение флага `HEAP_NO_SERIALIZE`.

Отдельные кучи

```
LPVOID HeapReAlloc (HANDLE hHeap, DWORD dwFlags, LPVOID lpMem,  
    DWORD dwBytes);
```

- Параметр dwFlags может быть комбинацией значений HEAP_NO_SERIALIZE, HEAP_REALLOC_IN_PLACE_ONLY и HEAP_ZERO_MEMORY.
- Действие флага HEAP_REALLOC_IN_PLACE_ONLY в HeapReAlloc противоположно действию LMEM_MOVEABLE в LocalReAlloc.

Отдельные кучи

```
DWORD HeapSize (HANDLE hHeap, DWORD dwFlags, LPCVOID lpMem);
```

- Флаг может иметь значение
HEAP_NO_SERIALIZE.

- Для удаления кучи используется вызов

```
BOOL HeapDestroy (HANDLE hHeap);
```

- Получить дескриптор локальной кучи можно с
ПОМОЩЬЮ

```
HANDLE GetProcessHeap (VOID);
```

Хранение данных

- Для постоянного хранения данных Windows CE реализует основанную на RAM файловую систему, известную как хранилище объектов (object store). В хранилище объектов располагаются файлы и реестр системы, а также базы данных Windows CE. Некоторые элементы хранилища объектов могут размещаться в ROM, при этом они не выделяются для пользователя ничем, кроме невозможности их изменения.

Хранение данных

- В дополнение к хранилищу объектов, которое является устройством по умолчанию, поддерживается работа с внешними накопителями и файловыми системами на картах памяти и других устройствах.
- Работа с хранилищем объектов реализуется посредством стандартных API Win32 для работы с файлами и реестром.

Хранение данных

- API для работы с базами данных уникален для Windows CE. Функции базы данных обеспечивают простой инструмент для управления и организации данных. Хотя они не сравнятся с базами SQL, API достаточно удобен для хранения и организации простых наборов данных, таких как списки адресов.

Файловая система

- Файловая система по умолчанию на всех платформах Windows CE – хранилище объектов. В дополнение Windows CE поддерживает до 256 различных устройств хранения. Windows CE не использует буквы дисков, как это принято на PC. Вместо этого, каждое устройство представлено подкаталогом в корневом каталоге. Традиционно, имя такого подкаталога «Storage Card». Если подключено более одного устройства, дополнительные устройства нумеруются «Storage Card 1», «Storage Card 2», и т.д.

Файловая система

- Формат имени файла совпадает с таковым для настольных Windows, поддерживаются длинные имена файлов. Полное имя файла может иметь длину до MAX_PATH, обычно 260 байт. Атрибуты файлов Windows CE включают стандартные read-only, system, hidden, compressed, archive.
- Понятие текущего каталога отсутствует в Windows CE, файлы определяются их полным именем.

Файловая система

- Стандартный набор операций ввода-вывода включает функции открытия, чтения, записи и закрытия файла.

```
HANDLE CreateFile (LPCTSTR lpFileName, DWORD dwDesiredAccess,  
    DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
    DWORD dwCreationDistribution, DWORD dwFlagsAndAttributes,  
    HANDLE hTemplateFile);
```


Файловая система

- Параметры этой функции включают:
 - полное имя файла
 - желаемый режим доступа (GENERIC_READ, GENERIC_WRITE)
 - режим разделения доступа (FILE_SHARE_READ, FILE_SHARE_WRITE)
 - атрибуты безопасности (должно быть NULL)
 - параметры открытия
 - ✓ CREATE_NEW Создать новый файл.
 - ✓ CREATE_ALWAYS Создать или очистить существующий.
 - ✓ OPEN_EXISTING Открыть существующий файл.
 - ✓ OPEN_ALWAYS Открыть существующий или создать новый.
 - ✓ TRUNCATE_EXISTING Очистить существующий файл.

Файловая система

- флаги и атрибуты файла
 - ✓ FILE_ATTRIBUTE_NORMAL
 - ✓ FILE_ATTRIBUTE_READONLY
 - ✓ FILE_ATTRIBUTE_ARCHIVE
 - ✓ FILE_ATTRIBUTE_SYSTEM
 - ✓ FILE_ATTRIBUTE_HIDDEN
 - ✓ FILE_FLAG_WRITE_THROUGH
 - ✓ FILE_FLAG_RANDOM_ACCESS
- последний параметр игнорируется и должен быть 0.
- Функция возвращает дескриптор открытого файла или `INVALID_HANDLE_VALUE` в случае неудачи.

Файловая система

- Для чтения и записи используются функции

```
BOOL ReadFile (HANDLE hFile, LPVOID lpBuffer,  
               DWORD nNumberOfBytesToRead,  
               LPDWORD lpNumberOfBytesRead, LPOVERLAPPED  
               lpOverlapped);
```

```
BOOL WriteFile (HANDLE hFile, LPCVOID lpBuffer,  
               DWORD nNumberOfBytesToWrite,  
               LPDWORD lpNumberOfBytesWritten,  
               LPOVERLAPPED lpOverlapped);
```

- Последний параметр должен быть NULL.

Файловая система

- Для изменения текущей позиции в файле используется

```
DWORD SetFilePointer (HANDLE hFile, LONG lDistanceToMove,  
                     PLONG lpDistanceToMoveHigh, DWORD dwMoveMethod);
```

- Обратите внимание на использование указателя на long для старших байт смещения. На практике обычно используется NULL (файлы до 4 Гб).
- Последний параметр определяет тип смещения: FILE_BEGIN, FILE_CURRENT или FILE_END.
- Для определения текущей позиции можно применить конструкцию

```
nCurrFilePtr = SetFilePointer (hFile, 0, NULL, FILE_CURRENT);
```

Файловая система

- Закрывать файл позволяет функция

```
BOOL CloseHandle (HANDLE hObject);
```

- Дополнительную функциональность предоставляют функции

```
BOOL SetEndOfFile (HANDLE hFile);
```

```
WINBASEAPI BOOL WINAPI FlushFileBuffers (HANDLE hFile);
```

```
DWORD GetFileSize (HANDLE hFile, LPDWORD lpFileSizeHigh);
```

- Кроме того, предусмотрена возможность отображения файлов в память.

Реестр

- Реестр Windows CE содержит данные о приложениях, драйверах, настройках и другую информацию о конфигурации. Реестр организован в иерархическую систему ключей и значений.
- Ключ может содержать как другие ключи, так и значения и по функциональности примерно соответствует каталогу файловой системы.

Реестр

- Windows CE поддерживает четыре корневых ключа:
 - HKEY_LOCAL_MACHINE Конфигурация оборудования и драйверов
 - HKEY_CURRENT_USER Настройки текущего пользователя
 - HKEY_CLASSES_ROOT Данные о соответствии OLE и типов файлов
 - HKEY_USERS Данные всех пользователей
- Пример использования ключей:
HKEY_LOCAL_MACHINE\Software\Microsoft\FreeCell

Реестр

- Значение – примитивный набор данных, хранимых в реестре. Значение может быть различных типов, включая строковые и бинарные. Каждое значение имеет имя и соответствующие данные.
- Реестр накладывает следующие ограничения на данные:
 - Длина имени ключа или значения – до 255 символов.
 - Размер данных значения – до 4 Кб.
 - Глубина иерархии – до 16 вложенных уровней.

Реестр

- Открыть ключ для работы позволяет функция

```
LONG RegOpenKeyEx (HKEY hKey, LPCWSTR lpszSubKey, DWORD  
ulOptions, REGSAM samDesired, PHKEY phkResult);
```

- Первый параметр указывает уже открытый ключ или константу для одного из корневых ключей, второй – имя подключа, в т.ч. сложное (например, “Software\Microsoft\Pocket Word”). Третий и четвертый параметры игнорируются и должны быть 0 и NULL соответственно. Последний параметр – указатель на переменную для дескриптора ключа. Функция возвращает ERROR_SUCCESS в случае успеха или код ошибки.

Реестр

- Чтение значений осуществляется вызовом

```
LONG RegQueryValueEx (HKEY hKey, LPCWSTR lpszValueName,  
                      LPDWORD lpReserved, LPDWORD lpType,  
                      LPBYTE lpData, LPDWORD lpcbData);
```

- Параметры задают дескриптор ключа, имя значения и указатели на буферы для типа данных, самих данных (может быть NULL для определения только типа и длины) и длины данных.

- Запись значения осуществляется похожей функцией

```
LONG RegSetValueEx (HKEY hKey, LPCWSTR lpszValueName, DWORD  
Reserved, DWORD dwType, const BYTE *lpData, DWORD cbData);
```

Реестр

- Тип значения используется только как вспомогательная информация при чтении и не влияет на способ сохранения данных. Может быть следующим:
 - REG_SZ Unicode-строка (оканчивается символом 0)
 - REG_EXPAND_SZ Unicode-строка с переменными окружения
 - REG_MULTI_SZ Набор Unicode-строк, оканчивается двумя символами 0
 - REG_DWORD
 - REG_BINARY Двоичные данные произвольного вида
 - и др.

Реестр

- Для удаления ключей и значений используются функции

```
LONG RegDeleteKey (HKEY hKey, LPCWSTR lpszSubKey);
```

```
LONG RegDeleteValue (HKEY hKey, LPCWSTR lpszValueName);
```

- **Закрывать используемый ключ можно вызовом**

```
LONG RegCloseKey (HKEY hKey);
```

Реестр

- Для получения всех подключей для данного ключа можно использовать

```
LONG RegEnumKeyEx (HKEY hKey, DWORD dwIndex, LPWSTR lpszName,  
                  LPDWORD lpcchName, LPDWORD lpReserved,  
                  LPWSTR lpszClass, LPDWORD lpcchClass,  
                  PFILETIME lpftLastWriteTime);
```

- Функция по дескриптору ключа и индексу (от 0) определяет имя подключа, имя класса и время последней модификации (не реализовано в Windows CE). Если ключа с заданным индексом нет, функция возвращает значение ERROR_NO_MORE_ITEMS.

Реестр

- Аналогичная функция предусмотрена и для перебора всех значений в ключе

```
LONG RegEnumValue (HKEY hKey, DWORD dwIndex, LPWSTR  
    lpszValueName, PDWORD lpcchValueName, LPDWORD lpReserved,  
    LPDWORD lpType, LPBYTE lpData, LPDWORD lpcbData);
```

- Параметры соответствуют параметрам RegQueryValueEx.

Нижегородский государственный университет им.Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Лаборатория: Математические и
программные технологии для
современных компьютерных систем
(Информационные технологии, ИТЛаб)

Лаборатория программного
обеспечения мобильных средств
связи (ЛМСС)

Разработка ПО для портативных компьютеров (КПК)

PocketPC (4)

Жерздев С.В.

Сетевые соединения

- WinSock в Windows CE обеспечивает как стек протоколов TCP/IP, так и стек протоколов для инфракрасного порта IrDA. Windows CE реализует подмножество API WinSock версии 1.1. В основном исключению подверглись функции асинхронной работы с сетевыми соединениями, использование которых можно избежать в многопоточных реализациях приложений.
- Тем не менее, можно соединение можно перевести в неблокирующий режим, так что каждый вызов будет возвращать, по необходимости, код незавершенной операции.

Инициализация WinSock

- Перед использованием библиотеки WinSock, ее необходимо инициализировать. Это можно осуществить вызовом
- `int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSADATA);`
- Функция возвращает 0 в случае успеха или код ошибки. В последнем случае не следует вызывать любые функции WinSock, в том числе `WSAGetLastError`.
- Первый параметр указывает необходимый номер версии библиотеки. Для указания этого параметра можно использовать макрос `MAKEWORD(1,1)`.

Инициализация WinSock

- Второй параметр – указатель на структуру, которая будет заполнена функцией

```
struct WSADATA {  
    WORD wVersion;  
    WORD wHighVersion;  
    char szDescription[WSADESCRIPTION_LEN+1];  
    char szSystemStatus[WSASYSSTATUS_LEN+1];  
    unsigned short iMaxSockets;  
    unsigned short iMaxUdpDg;  
    char FAR * lpVendorInfo;  
};
```

- Из полезных полей этой структуры можно отметить iMaxSockets – максимальное число соединений, и iMaxUdpDg – максимальный размер датаграммы.

Инициализация WinSock

- Для завершения работы предусмотрена функция

```
int WSACleanup ();
```

- В Windows CE она не выполняет никаких действий, но оставлена для совместимости.
- Практически все строковые поля в структурах, используемых для работы с сетью, являются не Unicode-строками.
- По этой причине будут полезны функции

```
int WideCharToMultiByte(UINT CodePage, DWORD dwFlags, LPCWSTR  
    lpWideCharStr, int cchWideChar, LPSTR lpMultiByteStr, int  
    cchMultiByte, LPCSTR lpDefaultChar, LPBOOL lpUsedDefaultChar);  
int MultiByteToWideChar (UINT CodePage, DWORD dwFlags, LPCSTR  
    lpMultiByteStr, int cchMultiByte, LPWSTR lpWideCharStr, int  
    cchWideChar);
```

Потоковые соединения

- Общая схема установки соединения со стороны сервера выглядит следующим образом
- 1. Создать socket
- 2. Привязать socket к адресу
- 3. Ждать запрос на соединение
- 4. Принять запрос на соединение
- 5. Принять/передать данные
- 6. Заккрыть соединение
- 7. Заккрыть socket
- Пункты 3-6 могут выполняться многократно при обработке множественных соединений.

Потоковые соединения

- Общая схема для клиента:
- 1. Создать socket
- 2. Установить соединение с сервером
- 3. Принять/передать данные
- 4. Заккрыть соединение

Потоковые соединения

- Для создания socket используется функция
`SOCKET socket (int af, int type, int protocol);`
- Первый параметр определяет вид адресации и в Windows CE может принимать одно из двух значений: `AF_INET` или `AF_IRDA`. Вторым параметром указывается тип соединения и может быть `SOCK_STREAM` или `SOCK_DGRAM`. Для IrDA поддерживается только тип `SOCK_STREAM`. Параметр `protocol` определяет протокол соединения. Функция возвращает `socket` или `INVALID_SOCKET` в случае ошибки.

Потоковые соединения

- Для привязки socket к определенному сетевому адресу используется

```
int bind (SOCKET s, const struct sockaddr FAR *addr, int  
         namelen);
```

- Второй параметр этой функции зависит от используемого протокола, а третий – определяет размер данных.

Потоковые соединения

- Для стандартных TCP/IP соединений это – указатель на структуру SOCKADDR_IN и ее размер:

```
struct sockaddr_in {  
    short sin_family;  
    unsigned short sin_port;  
    IN_ADDR sin_addr;  
    char sin_zero[8];  
};
```

- Первое поле должно быть установлено в AF_INET. Второе поле определяет IP порт, а третье – IP адрес. Последнее поле не используется.

Потоковые соединения

- Для соединений IrSock используется структура SOCKADDR_IRDA:

```
struct sockaddr_irda {  
    u_short irdaAddressFamily;  
    u_char irdaDeviceID[4];  
    char irdaServiceName[25];  
};
```

- Поля должны содержать AF_IRDA, 0 для сервера и строку-идентификатор сервера соответственно.

Потоковые соединения

- После создания socket, привязки его к адресу, сервер переводит его в режим ожидания запросов.

```
int listen (SOCKET s, int backlog);
```

- Второй параметр – максимальный размер очереди ожидающих запросов. Для Windows CE поддерживаются только значения 1 и 2.

Потоковые соединения

- Вызов функции

```
SOCKET accept (SOCKET s, struct sockaddr FAR *addr, int FAR  
*addrlen);
```

- блокирует до получения запроса от клиента на установку соединения. Второй и третий параметр характеризуют структуру для адреса клиента. В Windows CE, в зависимости от типа соединения, это может быть SOCKADDR_IN или SOCKADDR_IRDA. Функция возвращает новый socket для данного соединения, исходный остается в режиме ожидания. В случае ошибки функция возвращает INVALID_SOCKET.

Потоковые соединения

- С клиентской стороны для инициации соединения используется

```
int connect (SOCKET s, const struct sockaddr FAR *name, int  
            namelen);
```

- В случае успеха функция возвращает 0.
- Прием и передача данных производится клиентом и сервером симметрично и независимо. Для этого используются функции

```
int send (SOCKET s, const char FAR *buf, int len, int flags);  
int recv (SOCKET s, char FAR *buf, int len, int flags);
```

- Последний параметр должен быть 0 или, для функции получения данных, может быть MSG_PEEK для извлечения данных без удаления их из очереди.

Потоковые соединения

- Для закрытия соединения используется вызов

```
int shutdown (SOCKET s, int how);
```

- Второй параметр определяет способ закрытия и может принимать значения SD_RECEIVE, SD_SEND или SD_BOTH для отказа от приема, передачи или обоих видов. Закрыть socket можно **ВЫЗОВОМ**

```
int closesocket (SOCKET s);
```