Мир ПК

19.03.2008г.

Мир ПК №02, 2008

Программирование простейших USB-устройств на Delphi

Михаил Перов

Что только нынче не подключают к компьютеру! Помимо традиционных клавиатуры, мыши и принтера можно встретить ключи защиты данных, кофейные кружки с подогревом, нестандартные преобразователи данных, всевозможные датчики, контроллеры и многое другое.

Но ведь мало только физически подсоединить устройство к компьютеру, нужно еще и наладить обмен данными между ними. Как же выбрать порт и организовать подключение? Несколько лет назад стандартным решением было использование COM-порта. Кстати, до сих пор различные специалисты доустанавливают на промышленные компьютеры по 8, по 16, а то и по 32 COM-порта (есть целая категория различных PCI-плат расширения последовательных портов, контроллеров и т. д.). Таким образом, если нужно подключить несколько внешних устройств с интерфейсом RS-232, могут потребоваться дорогие адаптеры и экзотические платы расширения, которые по старой традиции неделями плывут в Россию на пароходах. Кстати, название обычного переходника «адаптер DB9m/DB25f» у менеджера компьютерного магазина может вызвать разве что раздражение.

Что такое HID-устройство

Сейчас практически все устройства подключаются к компьютеру через USB-интерфейс. Поэтому во многих новых ПК COM-порт отсутствует вообще.

USB-интерфейс — типовое решение по сопряжению нового внешнего устройства с компьютером, точнее, это HID-интерфейс, базирующийся на протоколе USB 1.1.

Хотя многие и считают, что HID-интерфейс (Human Interface Device) предназначен исключительно для клавиатуры, мыши и джойстика, он годится для множества решений, связанных с сопряжением внешних устройств и компьютера.

Если пользователю необходимо производить низкоскоростной обмен данными (до 64 кбит/с) и при этом желательно сократить время на утомительной разработке собственных драйверов, то ему вполне подойдет HID. На выходе же получится простое и вполне современное решение на базе стандартного программного USB-интерфейса с гарантированной поддержкой на всех распространенных программных платформах.

Свойства HID-устройства

С точки зрения организации программной поддержки HID-устройства, все выглядит достаточно привлекательно: для работы под управлением Windows можно быстро создавать понятный компактный код на базе готовых проверенных алгоритмов. При этом у разработчика останется масса времени на реализацию собственного протокола обмена данными верхнего уровня, поскольку необходимый уровень абстрагирования уже организован за счет HID-протокола (см. таблицу). Кроме того, программисту легко проводить отладку написанного протокола обмена (разумеется, при наличии работающего HID-устройства) — благодаря относительной жесткости самого протокола достаточно просто разработать программу поддержки устройства компьютером. Еще бы! Массу работы уже взял на себя создатель HID-устройства.

Параметр протокола обмена для HID-устройства	Значение параметра	Примечание
Режим передачи данных с низкой скоростью	800 байт/с	Максимально возможная скорость реального COM-порта — до 32 000 байт/с (на практике меньше)
Режим передачи данных с высокой скоростью	60 000 байт/с	
Максимальный размер репорта[1] (структуры обмена данными)	65 535 байт	
Поддержка на уровне операционных систем	Для Windows XP: имеет встроенные HID-драйверы Для Linux: стабильная поддержка начиная с ядра 2.4.2	
Структуры приема-передачи данных	Входной репорт (Input)	
	Выходной репорт (Output)	
	Специальный репорт (Feature)	Используется при критичном времени доставки данных

[1]Репорт — структура данных для обмена информацией с HID-устройством. Термин является устоявшимся в среде программистов USB-устройств

Организация обмена данными между HID-устройством и компьютером

Чтобы описать взаимодействие HID-устройства с компьютером, употребим термин «хост». В данном случае под ним понимается управляющее устройство в общей физической архитектуре взаимодействия по USB-протоколу. Так, все порты в компьютере — хосты. К ним можно подключать различные USB-устройства (флэшки, мыши, веб-камеры, фотоаппараты и проч.), которые хоста не имеют. Хост обеспечивает обнаружение, подключение, отключение, конфигурирование устройств, а также сбор статистики и управление энергопотреблением.

HID-устройство может само установить частоту опроса, во время которого выясняется наличие в нем каких-либо новых данных. Значит, даже на таком низком уровне программист может довериться системе, поскольку частота опроса и другие параметры обмена данными должны быть заранее заданы в программе контроллера HID-устройства. Этим протокол HID отличается от общего описания USB 1.1 или USB 2.0, в котором нет жестких требований к организации протокола. Однако при специфических задачах, требующих повышенного уровня безопасности, может оказаться довольно сложно избавиться от циклических опросов, когда постоянно передаются почти одни и те же блоки данных.

Особенности программирования HID-устройств

HID-устройства имеют специальные дескрипторы. Когда хост определит, что устройство принадлежит к классу HID, он передает управление им соответствующему драйверу. Предполагается, что дальнейший обмен данными ведется под его руководством.

В Windows за доступ к HID-устройствам отвечает системная служба HidServ. Подробнее о функциях запросов к HID-устройствам и других особенностях работы с HID-драйвером рассказывается в работе П.□В. Агурова «Интерфейс USB. Практика использования и программирования» (СПб.: БХВ-Петербург, 2005).

Программирование HID-устройств на «верхнем уровне»

Нелегкую жизнь «прикладных» программистов, работающих на Паскале, облегчает проверенный модуль HID.PAS, программная оболочка для hid. dll (Hid User Library — как указано в свойствах файла). В комментариях к файлу сообщается, что в основе его лежат модули hidsdi.h и hidpi.h корпорации Microsoft. А сам файл HID.PAS — часть пакета JEDI (http://jvcl.sourceforge.net).

Для работы с HID-устройством в среде Delphi for win32 применяется компонент TJvHidDeviceController, представляющий собой удобный глобальный менеджер для доступа к HID-устройствам. А уже на его базе можно получить объектный экземпляр для работы с конкретным устройством.

Основные свойства и события компонента TJvHidDeviceController

Рассмотрим компонент TJvHidDeviceController более подробно. Событие OnArrival срабатывает на поступление (подключение) в систему HID-устройства, доступ к устройству предоставляется в обработчике этого события через экземпляр класса TJvHidDevice. Простое событие OnDeviceChange реагирует на изменение состояния

устройства, оно только сигнализирует об изменениях в системе. Событие OnDeviceData срабатывает при поступлении данных от одного из HID-устройств и передает обработчику следующее: HidDev: TJvHidDevice; — устрой-ство, от которого были получены данные;

```
  ReportID: Byte; – номер дескриптора репорта;
  Data: Pointer; Size: Word; –
параметры для получения данных.
```

Событие OnDeviceDataError уведомляет об ошибке передачи данных, передавая в процедуру обработки параметры HidDev: TJvHidDevice; — HID-устройство и Error: DWORD; — код ошибки. Событие OnDeviceUnplug уведомляет об извлечении устройства из списка установленных в системе. Типы обработчиков событий на Plug и Unplug одинаковы (в исходном тексте: TJvHidUnplugEvent = TJvHidPlugEvent). В обработчик передается объект класса TJvHidDevice, соответствующий HID-устройству.

Для последовательного перечисления имеющихся в системе HID-устройств по вызову метода Enumerate предназначено событие OnEnumerate, т. е. в обработчике события найденные устройства последовательно передаются в виде объектов. Это событие принудительно инициируется методом Enumerate, использующимся для «проведения» имеющихся HID-устройств через обработчик, например при ревизии состояния HID-устройств по инициативе хоста (компьютера).

Событие OnRemoval срабатывает на физическое извлечение устройства из системы и имеет тот же тип обработчика TJvHidUnplugEvent, что и для OnDeviceUnplug. Функция CountByProductName выдает количество устройств, удовлетворяющих указанному в аргументе имени продукта, а CountByVendorName — указанному в аргументе имени производителя.

Основные свойства и события класса TJvHidDevice

Класс TJvHidDevice — виртуальное представление отдельно взятого HID-устройства. Новый объект этого класса можно получить, как было уже сказано, из события OnArrival или OnEnumerate. Функционал классов TJvHidDeviceController и TJvHidDevice частично дублируется, поскольку в первом из них интегрированы общий инструментарий для работы с набором имеющихся в системе HID-устройств и механизм доступа к одному из них. Устройство можно однозначно идентифицировать по свойствам SerialNumber, ProductName и VendorName. Чтобы получить сведения о поступлении данных с применением такого объекта, можно воспользоваться событием OnData. Отсылка данных ведется через метод WriteFile (в строгом смысле — через функцию). WriteFile — это оболочка системной функции WriteFile (kernel32).

Чтобы проконтролировать факт извлечения устройства, следует присвоить свой обработчик событию OnUnplug. Перед началом обмена данными с HID-устройством нужно удостовериться в самой возможности такого обмена с помощью HasReadWriteAccess. В этом классе на возникновение ошибки обмена данными даже есть отдельное событие OnDataError.

А теперь рассмотрим фрагменты кода из «живого» проекта, реализующего тестовое клиентское приложение для организации обмена данными с нестандартным устройством — пластиковыми чип-картами на базе HID. В борьбе за реализм автор взял на себя смелость не выкидывать из листингов «лишние» технологические обвязки кода.

Метод ScanDevices (листинг 1) предназначен для инициирования процесса поиска в системе необходимого HID-устройства. Большая часть кода, за исключением вызова метода Enumerate, необязательна и обеспечивает гибкость приложения, например, для того, чтобы в эту же тестовую программу можно было добавить возможность работы по интерфейсу, отличному от HID. Метод AddError выводит в окно отладочную информацию в процессе работы программы.

Листинг 1. Инициирование поиска устройства

```
procedure TITCR2.ScanDevices;
begin
    if DriverComponent<>nil then begin
     if (DriverComponent as TComponent).
ClassName='TJvHidDeviceController'
     then begin
    (DriverComponent as
TJvHidDeviceController)
OnEnumerate:=HidCtlEnumerate;
  // Инициируем перебор имеющихся в системе
  // HID-устройств
    (DriverComponent as
TJvHidDeviceController).Enumerate;
     end else begin
      self.AddError('TITCR2.Init:
ClassName<>TJvHidDeviceController');
      Exit;
     end;
    end else begin
     self.AddError('TITCR2.Init:
DriverComponent = nil');
     Exit;
    end;
  end;
```

В листинге 2 приведен обработчик события OnEnumerate для поиска необходимого внешнего устройства. Для простоты будем считать, что программа может работать только с одним устройством нужного ей типа.

Листинг 2. Обработчик поиска

```
function TITCR2.HidCtlEnumerate(HidDev:
TJvHidDevice;
  const Idx: Integer): Boolean;
  var
   N: Integer;
   PN:String;
  begin
  PN:= Trim(HidDev.ProductName);
    if PN <> '' then begin
  // Пытаемся найти необходимое нам
  // оборудование
    if (PN = 'Card Reader UUS1')
     then begin
            ProductName_:= PN;
            SerialNumber_:=HidDev.SerialNumber;
            VendorName_:=HidDev.VendorName;
  // Принимаем это устройство как текущее
  // для дальнейшей работы
            Device_:=HidDev;
     end else begin
  //=========================================
  // Другие действия, если устройство
  // не распознано
  //=========================================
     end;
    end;
    Result:= True;
  end;
```

Прежде чем рассматривать дальнейшую реализацию проекта, следует немного рассказать о принятом формате обмена данными верхнего уровня, т. е. о структуре, призванной быть посредником между методами приема-передачи данных и конкретной решаемой прикладной задачей. Дело в том, что здесь разработчику предоставляется возможность реализовать свои творческие способности. Вернее, разработчикам, потому что процесс создания нового протокола очень часто бывает двусторонним, и при этом первую скрипку играет тот, кому труднее реализовывать алгоритм обмена. В общем, каким бы ни был протокол обмена, всегда приятно делать каждую программную сущность максимально наглядной и самодостаточной, пусть даже в ущерб некоторым общепринятым традициям. Ибо лучшее решение — то, которое будет реализовано в сжатые сроки с

минимальной привязкой к программной среде и с большими возможностями дальнейшего развития. На основе этих принципов был создан протокол обмена верхнего уровня, где главное понятие — «команда». Из листинга 3 видно, насколько автор любит строковые данные, не раз спасавшие его при отладке программных модулей. Как же замечательно, что у нас вообще есть тип String! Все команды протокола делятся на категории (классы), внутри которых существует код команды, однозначно характеризующий ее назначение. Параметр edParam служит для отсылки данных в устройство, а параметр edAnswerData содержит в себе полученные от устройства данные. Строковый тип описанных членов записи позволяет свободно и наглядно манипулировать данными в формате HEX-строки. И что самое приятное, формат описанной записи идеологически стоит где-то посередине между ее непосредственным назначением и различными формами ее представления (INI, HEX, XML и т. д.)

Листинг 3. Пример структуры отсылки данных для описания алгоритмов обмена данными с внешним устройством.

```
// Структура соответствует протоколу
// верхнего уровня,
// оговоренному для специфического устройства.
TedCommand = RECORD
 Caption:String;  // Метка команды
 INDEX:Integer;   // Индекс команды
 Enabled:Boolean; // Флаг пригодности команды
 WT:Cardinal;     // Время обработки команды
 edClass:String[2];// HEX-строка класса команды
 edCode:String[2];// HEX-строка кода команды
 edLength:String[2];     // Длина
 edParam:String;  // HEX-строка (параметр отсылки)
 edAnswerData:String;// HEX-строка
// полученных данных
 IsAnswerData:Boolean;// Флаг новых
// полученных данных
 ERROR:String;    // Поле для последней ошибки
 WORKED:Boolean;  // Флаг обработки команды
end;
```

Выполнение команды, т. е. отсылка данных в устройство, реализовано с применением отсылки пакетов данных длиной 8 байт (листинг 4). Эта длина — не единственное решение, такой выбор продиктован требованиями протокола верхнего уровня и в каждом конкретном случае может быть другим. Это, что называется, дело вкуса. Странный флаг IsUSBMode в методе ExecuteCommand (листинг 5 на «Мир ПК-диске») оставлен как напоминание о том, что вместо работы с USB нам может потребоваться использовать COM-порт или какой-то другой интерфейс. В начале отсылаемой группы данных в устройство передается синхросерия произвольно выбранного формата (например, 3E3E3E2B), сообщающая устройству, что у него на входе вполне легальные данные. Напомню, что в данном случае речь идет не столько о HID, сколько о специфическом протоколе верхнего уровня, идеологически оторванном от «железа» и предназначенном для решения особых прикладных задач.

Листинг 4. Отсылка данных

```
function TITCR2.ExecuteCommand(var edCommand:
TedCommand; const param1,
   param2: Integer): Integer;
var HD:TJvHidDevice;
    Buf: array [0..64] of Byte;
    Written: Cardinal;
    ToWrite: Cardinal;
    blockString,sblockString:String;
    bN,i:Integer;
begin
   result:=-1;
    try
    if not(edCommand.Enabled) then Exit;
    edCommand.WORKED:=false;
    edCommand.WT:=GetTickCount;
    if IsUSBMode then begin
     try
     LastCMD_:=edCommand;
  // Подготовка строки отсылки данных,
  // начальная последовательность байт
  // соответствует протоколу обмена
  // верхнего уровня, специфичному для
  // данного устройства
     blockString:=Trim('3E3E3E2B'+
        edCommand.edClass+
        edCommand.edCode+
        edCommand.edLength+edCommand.edParam);
      // Достраиваем блок
      bN:=Length(Trim(blockString)) div 16;
      if ((Length(Trim(blockString)) / 16) -
  bN)>0 then bN:=bN+1;
       while true do begin
        if Length(Trim(blockString))>=(bN*16)
then break;
       blockString:=blockString+'00';
      end;
     // Отсылаем данные по 8 байт
     for i:=0 to bN-1 do begin
      sblockString:=Copy(blockString,i*16,16);
      SendData8b(sblockString);
     end;
      LastCMDTime_:=Now;
      edCommand.WORKED:=true;
      edCommand.WT:=GetTickCount-edCommand.WT;
      except on e:Exception do begin
      result:=-3;
      //=====================================
      // Обработка ошибки
      //=====================================
      end;
     end;
     end;
     except on e:Exception do begin
       result:=-2;
      //=====================================
      // Обработка ошибки
      //=====================================
       end;
    end;
  end;
```

В обработчике GetDataExecutor полученных от устройства данных (пакет по 8 байт) использовано специально созданное событие OnNewInputData для передачи первично обработанных данных на дальнейшую обработку, причем с указанием их старого и нового значений (листинг 6 на «Мир ПК-диске»). Таким образом, события поступления необработанных данных и указание на дальнейшую обработку развязываются, позволяя добавлять какой-то специфический алгоритм предупреждения на раннем этапе ошибочной, повторной или ненужной входной

информации.

Представленные здесь примеры работы с HID-устройством иллюстрируют общую идею статьи — относительную простоту программирования нестандартных HID-устройств средствами Delphi.
