

Straight Talk About USB

Sure, we all know the advantages of USB for end users: plug-and-play recognition of your peripheral device by the PC, ability of the USB to power your device, and so on. Those are big plusses. What you may not know about are these benefits for developers:

- ◆ a simple, well defined electrical and mechanical interface
- ◆ a robust protocol that has already been debugged for you
- ◆ a way of talking to multiple devices on the same external bus
- ◆ a way of making your peripheral describe its own interface to software written by others (or to different versions of your own software)

Let's compare USB to something that developers are very familiar with: the RS-232 serial port. Even though the lowly serial port has been around for decades, people still have problems with it, to this very day.

For one thing, there's the wiring: all those wires to come loose or get miswired. How many times have you had problems with DTE vs DCE connector wiring? Did you get out the little thingy with the LEDs on it to help you figure out the problem? Did you then find out that your cable wouldn't connect to it, because it was the wrong gender or that you needed a 9-pin to 25 pin adapter???

By contrast, USB only has 4 wires, and two of those are Power and Ground. It's pretty easy to get things wired up right.

Once you get past the wiring problems, so that your device and your PC can send signals to each other, you get to the next level of the problem: you have to figure out what these two are gonna SAY to each other. In fancier words, you have to design a protocol. It takes careful thought to design that protocol right, and to make it cover all the things that can go wrong during operation.

Even after you've figured out in detail what the PC and the peripheral are supposed to say to each other under all circumstances, you still have to make both sides actually DO that, and do it correctly. This is a major source of headaches. Usually what happens is that you start writing code for one side, then you write code for the other side. No matter how carefully you follow your blueprint as you code, at some point you get interrupted: by a phone call, or lunch, or your boss, or whatever. This makes you forget at least one small but important detail on one side or the other.

When you think you're done with the programs for both sides, you compile them, program the peripheral-side code into the device, and power things up. Invariably, you find that the two sides are not talking to each other properly, and you get to spend days or even weeks trying to debug the problem. If you're lucky, the problem is big and obvious and shows itself right away. If you're not so lucky, the problem lurks in the background and only shows up under rare circumstances.

By contrast, USB's Human Interface Device class provides a capable, robust protocol that has already been debugged for you, and is not limited to mice and keyboards and joysticks. You can use it for all kinds of devices, like many of the data acquisition devices that people often design for RS-232.

It's obvious that USB can allow multiple peripherals to be connected to the same PC. (Try doing *that* with your serial port!) It may not be so obvious, but USB can even support the connection of multiple, identical devices to the same PC, even if those devices do not have unique serial numbers. HIDmaker Combo shows what you can do with those.

Finally, USB's Human Interface Device class provides a compact way for your peripheral device to actually describe itself to the PC. This system describes not only the name and manufacturer and model number of the device, but detailed information about each item of data that travels between the PC and the peripheral. Each item is described by how wide it is in bits, what range of values it may take, and even what it is used for. On top of that, the peripheral describes to the PC how it will pack the data items together in the most compact manner before it sends it to the PC, and it also describes the format it expects for data that is sent from the PC.

Using this system, your peripheral can be made to be usable by programs written by others, if you want. This sort of "open architecture" approach can help you create peripheral devices with a wider market. It can also help you write PC-side applications that can correctly handle older versions of your own device, to minimize maintenance headaches. (If you'd rather keep your interface more private, you can do that, too. We'll show you how.)

“But Isn’t USB Awfully Complicated?”

Sure it is. Part of that is because there are extra capabilities to manage, but mostly the problem is that the developer interface hasn’t been made very friendly up to now. If you’ve ever tried the traditional way of developing a USB-based project, you know there’s a serious amount of material you have to learn.

On the peripheral side, you need to understand a lot of arcane rules about writing and interconnecting descriptor tables. Learning to write Report Descriptors is like having to learn to program all over again, directly in binary. (Just what you needed, right?)

And on the PC side, you have to deal with a really complex Windows API, for which very few example programs are available. And because both the peripheral and the PC side have to work together, you need to get both sides exactly right, or nothing will work at all...

Yes, doing things the old way takes a steep learning curve for developers. It can take weeks to learn all that stuff, and bust your budget in the process. You either need to hire a USB expert, or you need the right tools...

USB As It Was Meant To Be

But now there’s a better way. With HIDmaker from Trace Systems Inc., you can literally get your project going in MINUTES. Just answer a few simple questions, describe the data that you want to send back and forth, and let HIDmaker figure out the rest.

HIDmaker will generate two working programs for you at the same time: one for the peripheral side, the other for the PC side. Compile these two programs with the compilers or assemblers that you already have, program the PicMicro in your target board, and you’re over the hump! You’ve created two programs that already know how to talk to each other correctly.

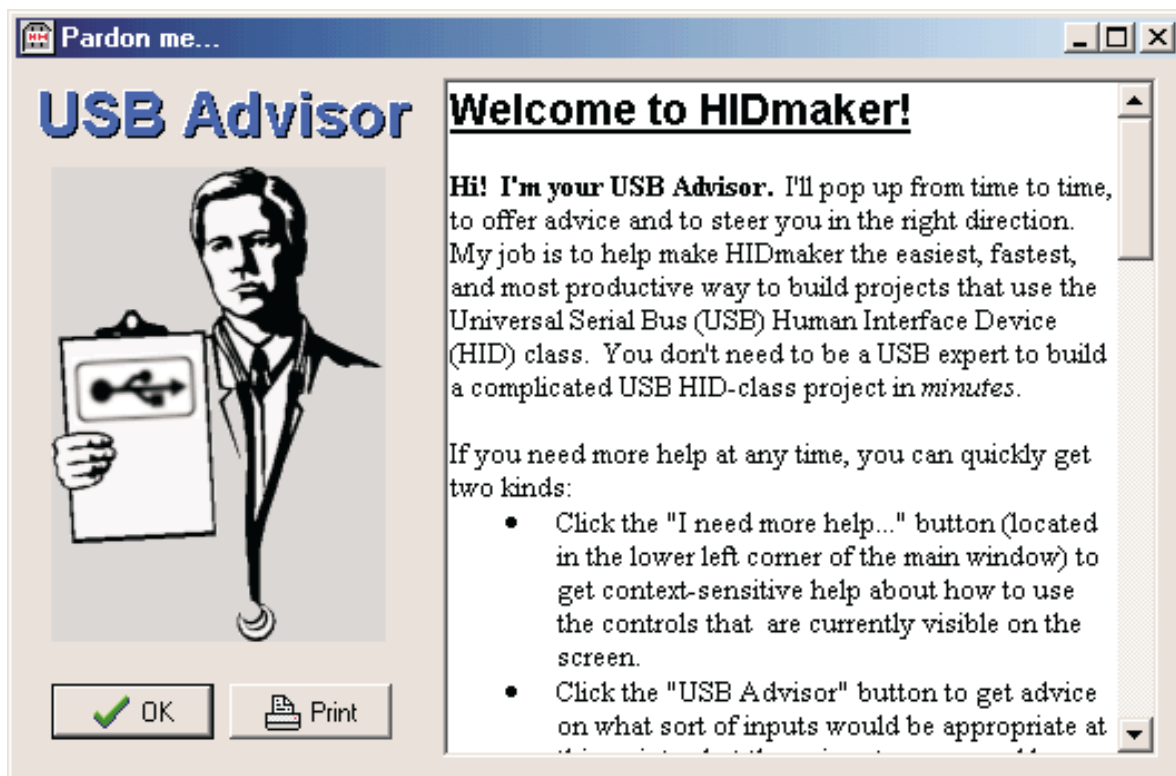
Automatically generating both the peripheral firmware and the PC-side program at the same time gives your project a solid starting point: two “known good” programs that can talk to each other correctly, ready for you to start customizing to make them do what you want.

Now you can spend your time doing what you do best: “personalizing” both sides to make them do what you want, without having to get bogged down in all the details of USB.

Here’s the best part. HIDmaker isn’t one of those tools that tries to “simplify” your job by merely preventing you from doing complex things. (You’ve seen the type. Tools like that are like a car whose steering has been “simplified” by only letting it make right turns. You can turn left by going around the block, but it takes you three right turns to do it! Some simplification...)

We found a way to truly simplify the creation of USB projects, without limiting your capabilities. We did that by completely re-thinking the whole process. Our approach is to ask the developer to supply only the absolutely essential information, and to let the HIDmaker program deal with all the rigid USB rules. (That's what computers are for, right?) That way, you won't have to learn, and constantly struggle to remember, a lot of complex stuff that really doesn't add much value.

HIDmaker Step By Step



✓ Step1: USB Advisor

It all starts by a chat with USB Advisor, your on-line USB expert. He understands what's most important to you: the specifics of what your project should do, not the details and rules of USB. With that in mind, he will ask you for only the bare essentials, and he will figure out the USB stuff for you, as much as possible.

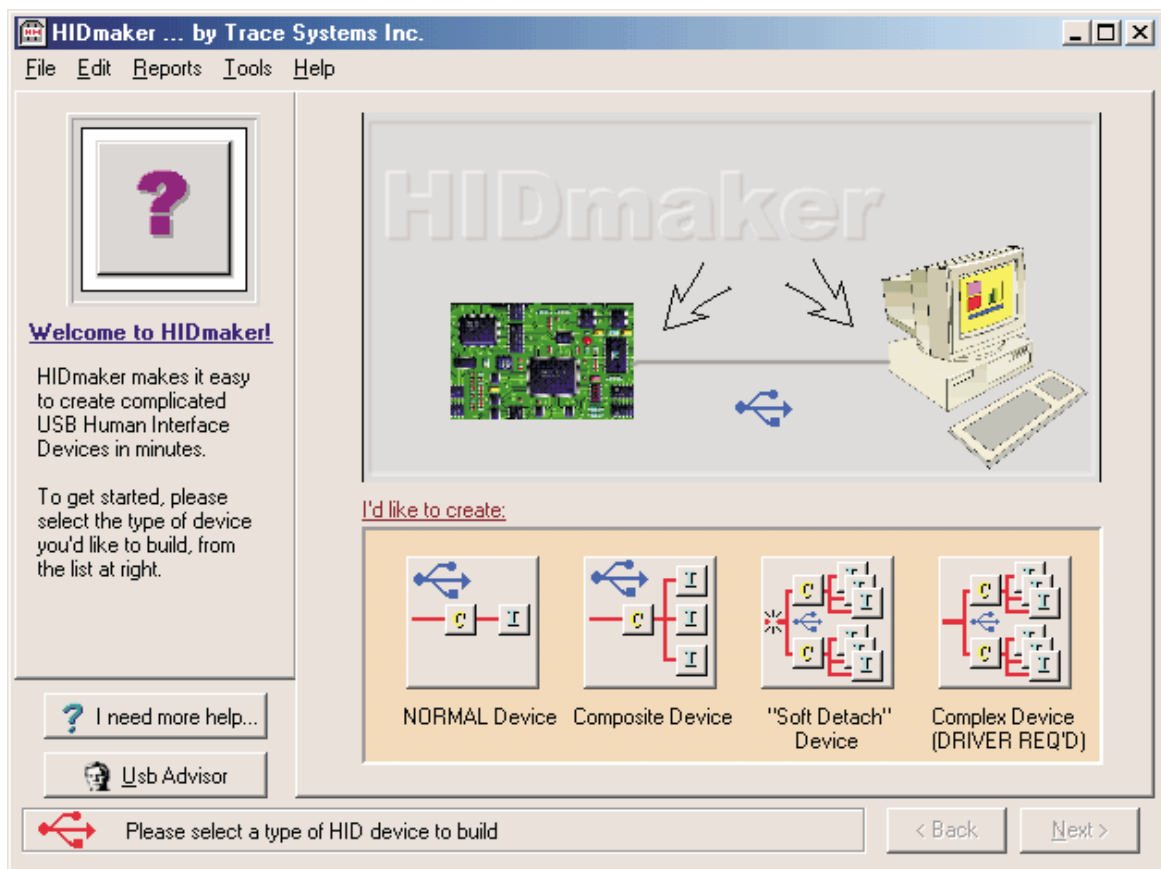
USB Advisor is always just a click away, ready to give you context-sensitive, useful advice in plain English. In fact, every screen in HIDmaker contains two buttons: "USB Advisor", and "I need more help...".

"USB Advisor" gives you insight into what's going on with the USB-related information you are currently being asked to provide, or tutorial information about USB and the Human Interface Device class.

The “I need more help...” button gives you context-sensitive information about the HIDmaker program itself: how to operate the controls on the current screen to accomplish the goal of this current step.

Both of these help buttons will provide you with a lot of useful information, especially the first time you use HIDmaker. If you don't finish your first USB project by lunchtime, don't worry. You can save your HIDmaker project and exit at any time, and then come back later to pick up where you left off. Once you get familiar with HIDmaker and USB, you'll be able to create a new project and generate a set of programs for it in minutes.

✓ Step 2: Select Your Device Type



USB devices can contain sections and sub-sections of capability. At any one time, a device may be in one particular USB “Configuration.” Each “Configuration” may contain several independent “Interfaces,” each of which acts like an independent thing. One example might be a device that contains a keyboard and a built-in pointing device. In one Configuration, the keyboard would be one Interface, and the pointing device would be another Interface, and would act like a mouse. Both Interfaces are independent and available at the same time. In another Configuration, the Interface for the pointing device might be changed to behave like a joystick.

Having multiple Configurations and Interfaces is flexible, but these things have to be correctly described in code or there will be big debugging headaches. This is another important detail that HIDmaker handles for you.

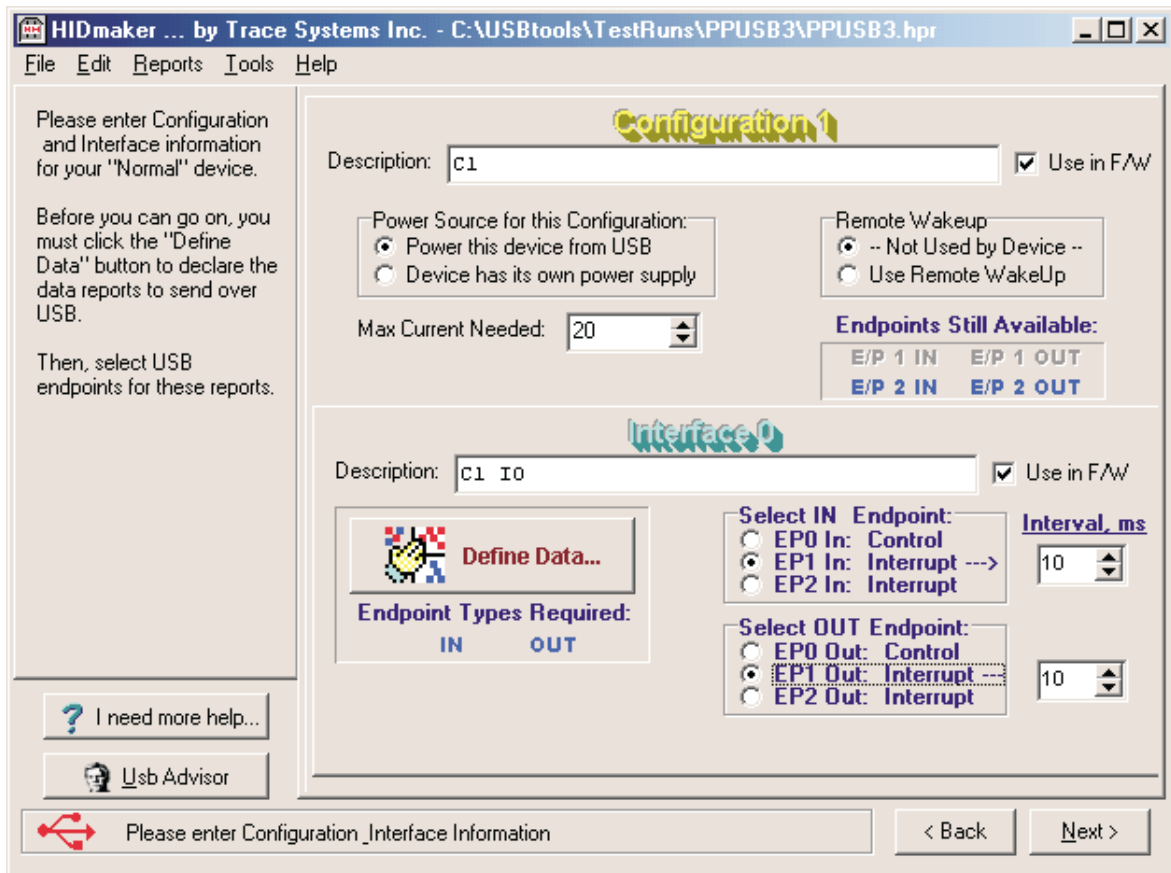
HIDmaker can create several types of USB Human Interface Device. Most devices will be of the “Normal” type, needing only a single Configuration and a single Interface. As the screen shot shows, you can select several other options, having more Interfaces and/or Configurations.

✓ Step 3: Enter Project Info

The screenshot shows the 'Project Information' dialog box in the HIDmaker application. The title bar reads 'HIDmaker ... by Trace Systems Inc. - C:\USBtools\TestRuns\PPUSB3\PPUSB3.hpr'. The menu bar includes 'File', 'Edit', 'Reports', 'Tools', and 'Help'. The main area is titled 'Project Information' with a note '(* = Required)'. The fields and their values are: Project File (C:\USBtools\TestRuns\PPUSB3\PPUSB3.hpr), Description (3rd PicProto USB Demo), Mfr. Name (Trace), Vendor ID (hex) (0925), Product ID (hex) (1006), Device Release No. (decimal) (1), Device S/N (V0.91), and Device Class Name (PPUSB3). Checkboxes for 'Use in F/W' are present for Description, Mfr. Name, Device Release No., and Device S/N. A sidebar on the left contains instructions: 'Use the Browse button to select a file name and location for this project.', 'Vendor ID and Project ID are required by Windows to identify your device. Devices sold commercially MUST obtain a Vendor ID from USB Implementers Forum.', and 'Optional fields can be placed in device firmware, by checking the nearby "Use in F/W" box. These strings should be kept SHORT.' At the bottom, there are buttons for 'I need more help...', 'Usb Advisor', 'Please enter general Project Information', '< Back', and 'Next >'.

Set the location for the project files that HIDmaker creates, and enter some information that describes your device. If you check the box next to a text item, that string will be placed in your device firmware in a way that can be read by the PC when your device operates.

✓ Step 4: Define Configuration and Interface Info



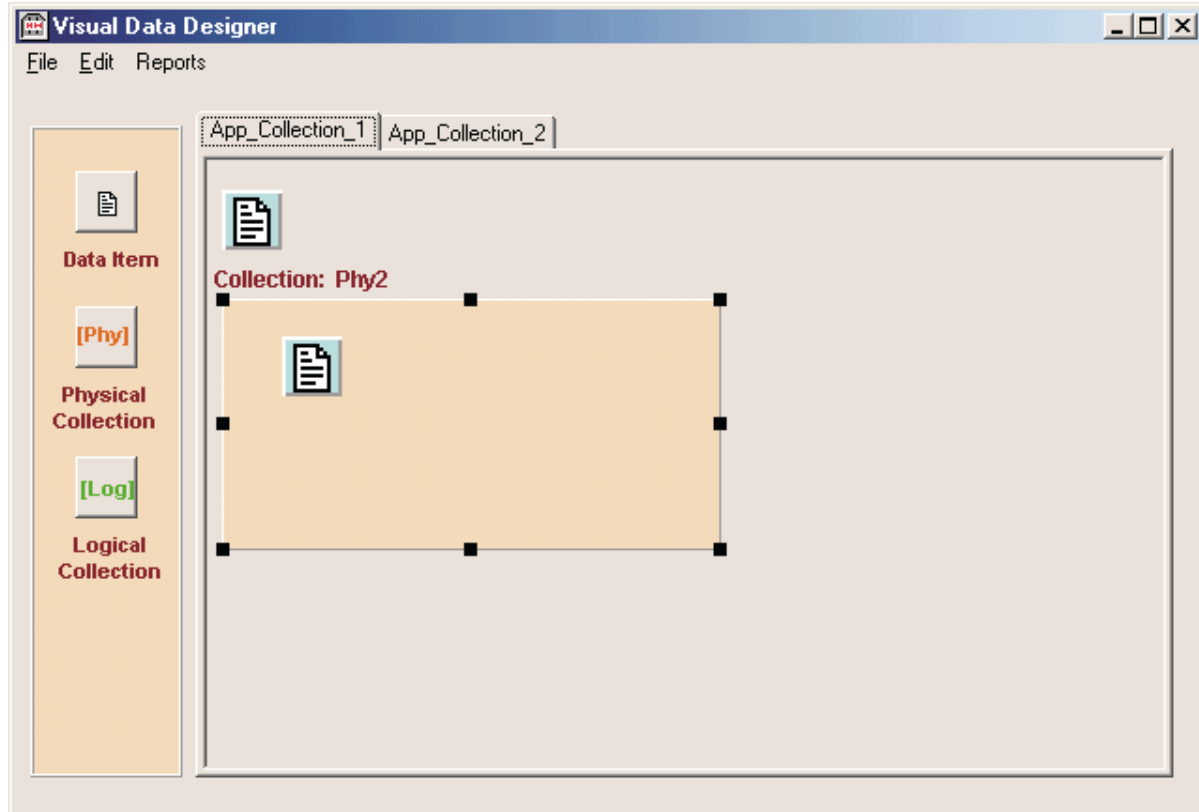
For each Configuration in your device, select a few options having to do with powering your device, and whether the device can be woken up from a low-power state by USB commands. For each Interface, set some information about the nature of the data (see next step) and which USB “Endpoints” this data will use.

✓ Step 5: Define Your Data -- Visual Data Designer

A key part of the project is your data. When you start out, you may not yet have a clear idea of all the data items you’ll need. That’s what the Visual Data Designer is for. Just click the button marked Define Data.

Visual Data Designer works somewhat like a smart sketch pad. You use it to graphically sketch the data items you will use, and to show the groupings of the data items (“Collections” in USB-speak).

If you’ve ever used a visual programming language like Visual Basic, Delphi, or C++ Builder, you’ll feel right at home. To add a new data item, you first click on a button on the Data Palette shown on the left, then you click on the drawing area on the right to create a data item of that type to your project. Items created directly on



the drawing area are part of the project's main group of data, which the USB Human Interface Device Class Specification calls the "Application Collection."

Sometimes you'll want to group data items together, perhaps because they correspond to some physical part of your device (like a group of buttons on a game pad), or perhaps because the data items belong together for some other logical reason. To do that, you first click either the Physical Collection or the Logical Collection button on the Data Palette, and click on the drawing area to add a panel that represents that collection. Go ahead and move it around or resize it if you want. Then, click one of the regular data items on the Data Palette, and then click on the collection panel you added to the drawing area. This will create a data item inside the collection.

You can continue this process as long as you like. You can add new data items on the main level, or inside the collection. You can add more collections, either onto the main level, or you can even create collections inside of collections if you want. As you go, you'll want to move things around on the drawing area to give yourself room.

To define the characteristics of each data item, just double-click on the item in the drawing area, and make your choices in the dialog box. For your convenience, HIDmaker will make some preliminary guesses for you, but you can always change them as you see fit.

Data Item Properties: Pot1

Required | Optional | Test Values

Required

Data Type: **Input** | Usage Info[0]: **0: Usage ID = 0x30, Usage Page = 0x1**

Data | Array | Absolute | No Wrap | Linear
 Constant | Variable | Relative | Wrap | Nonlinear

Preferred State | No Null State | Non-Volatile | Bit Field
 No Preferred State | Null State | Volatile | Buffered Bytes

Logical MINimum: **-128** | Logical MAXimum: **127** | Report Count: **1** | Report Size (Bits): **8**

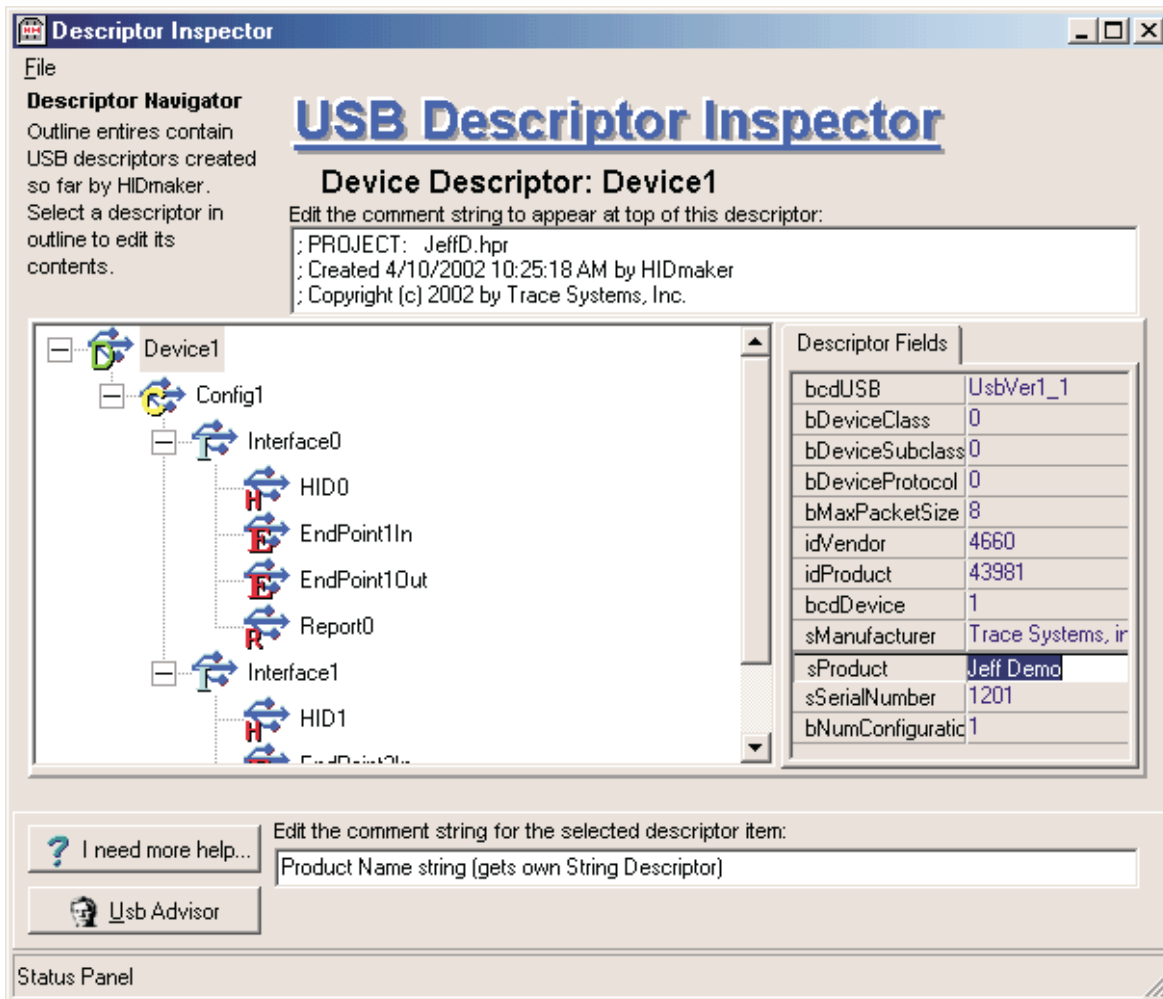
Name: **Pot1**

We can't go into a full explanation here of what all the settings mean, but you can find it in the manual and online help. (Most of the time, the default values are fine.) Most importantly, you'll find that the controls will interact in such a way as to prevent you from entering choices that don't make sense. For example, some choices in the screen shot are invalid for "array" items, so if you click "Array," the corresponding controls disable themselves. HIDmaker's visual environment always checks your work and will not let you make errors.

When you're satisfied with the data you've defined and are ready to go on, you can exit from the Visual Data Designer, saving the data definitions to a file if you want. This will take you back to HIDmaker's Configuration and Interfaces page. Labels will appear under the Define Data button that indicate what sort of USB Endpoints are needed, so check the boxes to dedicate the necessary Endpoints for this Interface, and you're ready to go on.

Clicking the Next button will cause HIDmaker to check everything you've entered so far, and if all is OK, to convert the information you've entered into standard USB Descriptors, including Device, Configuration, Interface, Endpoint, and Report descriptors. You're just about done.

✓ Optional Step: Descriptor Inspector



Once HIDmaker has generated USB descriptors, you're just about ready to generate code for the peripheral firmware and the PC-side application program. It's not necessary, but if you'd like, you can use the Descriptor Inspector item of the Reports menu to give you a last chance to look over the decisions that HIDmaker has made, and even modify some things if you feel confident in doing so.

On the Descriptor Inspector screen, you'll see a tree of USB descriptors on the left side, a grid of values on the right side, and several edit box controls.

In the tree on the left, you'll see the structure of USB descriptor tables that HIDmaker has created, based on the information you have provided so far. The outline tree structure shows the interrelationships between all these descriptor tables.

To inspect and possibly edit the values in one of these descriptor tables, you start by clicking on the item in the tree outline on the left, to select that descriptor. For example, suppose you click on Config1, a USB Configuration Descriptor. When you click on that descriptor, its editable values are shown in the inspector grid on the right side of the screen, the title changes to indicate that we have selected that descriptor table, and the edit box above the inspector grid fills with the text of a comment, describing this particular descriptor table, that will appear above this descriptor table in the final peripheral-side code. Feel free to modify this comment text to document this descriptor table correctly in your project.

If you've ever used a visual programming language like Visual Basic, Delphi, or C++ Builder, you'll know exactly what to do with the inspector grid on the right. It has two columns: the left column shows the name of an item, which cannot be changed, and the right column shows the value of the item, which can be changed. You can drag the left edge of the inspector grid to see more of the tree or the inspector grid. Likewise, you can drag the vertical centerline between the two columns of the inspector grid to get more room on either column of the grid.

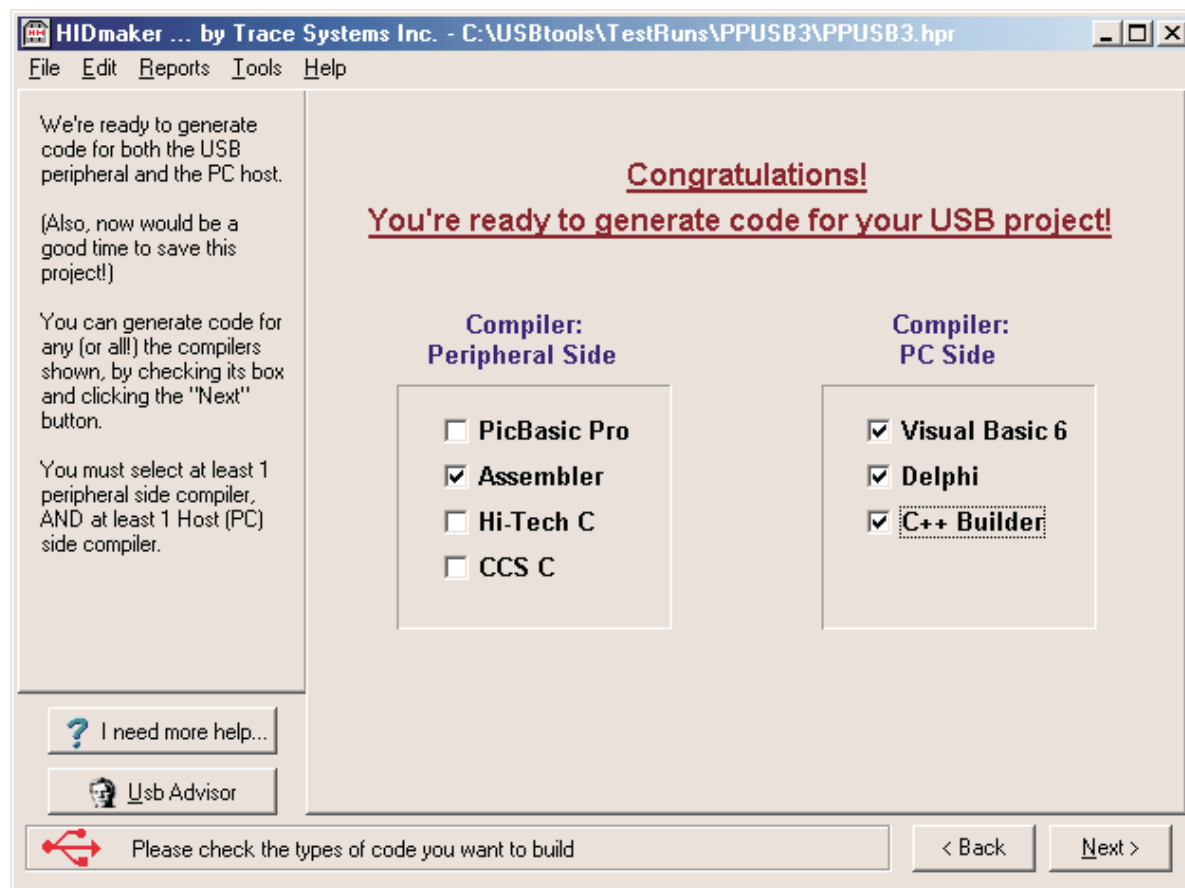
If you click on one of the lines in the inspector grid, it will highlight, and fill the edit box below the inspector grid with the text of a comment that describes this particular item in this descriptor table. This comment text will appear next to this item of this table, in the final peripheral-side code. Modify this comment as you like, to document this particular value correctly in your project. You will find that HIDmaker supplies default values and comments for all descriptor tables and values; you don't need to change any values or comments if these defaults are acceptable to you.

Sometimes, when you click on the value of an item, you'll see a drop down box so you can select a permissible value from a list. At other times, you'll see a small button containing three ellipsis dots '...'. Click that button to bring up a dialog box to edit the value.

✓ Step 6: Generate and Test the Code

When you're satisfied with all the descriptor tables, their values, and their comments, you're ready to generate code for both the peripheral and the PC-side programs. Save your work at this point, and click the "Next" button. HIDmaker will check all of your descriptors (including any new values or whole descriptor tables you have added) at this point, and advise you of any errors or changes you need to make.

When all problems have been resolved, HIDmaker will present you with programming language choices. On the left column, choose your favorite PICmicro programming language: PicBasic Pro, Assembler, Hi-Tech C, or CCS C, to let HIDmaker generate the firmware for your USB HID-class peripheral. On the right column, choose your favorite Windows programming language: Visual Basic,



Delphi, or C++ Builder. HIDmaker will generate a PC-side program that automatically knows how to talk to the peripheral it just generated.

Use the PIC language compiler that you already own to compile the peripheral side code that was generated by HIDmaker. Program the hex file produced by your compiler into a device, and install that device into your peripheral circuit board.

Similarly, use the PC-side language compiler that you already own to compile the PC-side application code that HIDmaker generated. Now, you're ready to test.

Plug the peripheral device into a USB cable connected to the device, and check with Control Panel to verify that the device enumerates properly. Once that happens, you are ready to run the PC-side application. The code that HIDmaker generates will create some fixed "dummy" data (with values you specified using Visual Data Designer), in a section of the code that is clearly marked by comments as being there for test purposes only. The PC-side application is a true GUI program that will allow you to send this dummy data back and forth between the PC and the peripheral, to verify that the USB connection works.

✓ Step 7: “Personalize” the Code

At this point, HIDmaker has done its job and has given your project a solid starting point: two “known good” programs that were created at the same time, and which know how to communicate with each other correctly. The USB part is done and working, so you can now concentrate on the things that you really set out to do in the first place: adding the real “personality” to your programs, on both the peripheral and the PC side, that makes your project do what you want it to.

We recommend that you make a copy of the original source code for both sides, then proceed to modify both of these programs, one step at a time. Add one feature or section of code, preferably on one side only, and test that new code carefully. We recommend that you get that feature working correctly before you move on. If something goes wrong, you’ll always know exactly where to look for the problem. That way, you’ll always have a working program at each step of the way, and things will never get out of control.

HIDmaker Software Framework

Extending the programs that HIDmaker generates is made easy by the HIDmaker Software Framework that the generated code follows.

In the peripheral side programs, a conveniently sized variable is created for each data item you define. For example, suppose you specified a data item named MyVar whose size is 5 bits. This 5 bit size is the number of bits that are transferred to or from the PC over the USB cable. To make your programming easier, the peripheral side program generated by HIDmaker will provide you with an 8-bit variable named MyVar. When you are ready to send this item from the peripheral to the PC, call the PackData routine to automatically extract the least significant 5 bits of MyVar and place these bits in the correct part of the correct packet of the USB HID report. An UnpackData routine is also available for unpacking data that comes from the PC to the peripheral.

The PC side programs all follow an object oriented approach that uses three main objects:

HIDagent, which does all the low level work of finding, opening, and accessing HID devices. HIDagent provides events that can tell your program when any HID device is attached or detached from the USB cables, opened or closed for I/O, or when a Report arrives or is selected. HIDagent also includes powerful methods such as PackAndSendReport() and ReadAndUnpackReport(), which handle all of the USB HID I/O from a single function call.

An object which represents one HID data item. The class name of this object is THidVar. It has properties such as Name, SizeInBits, UnscaledValue, ScaledValue, Logical Minimum, and so on.

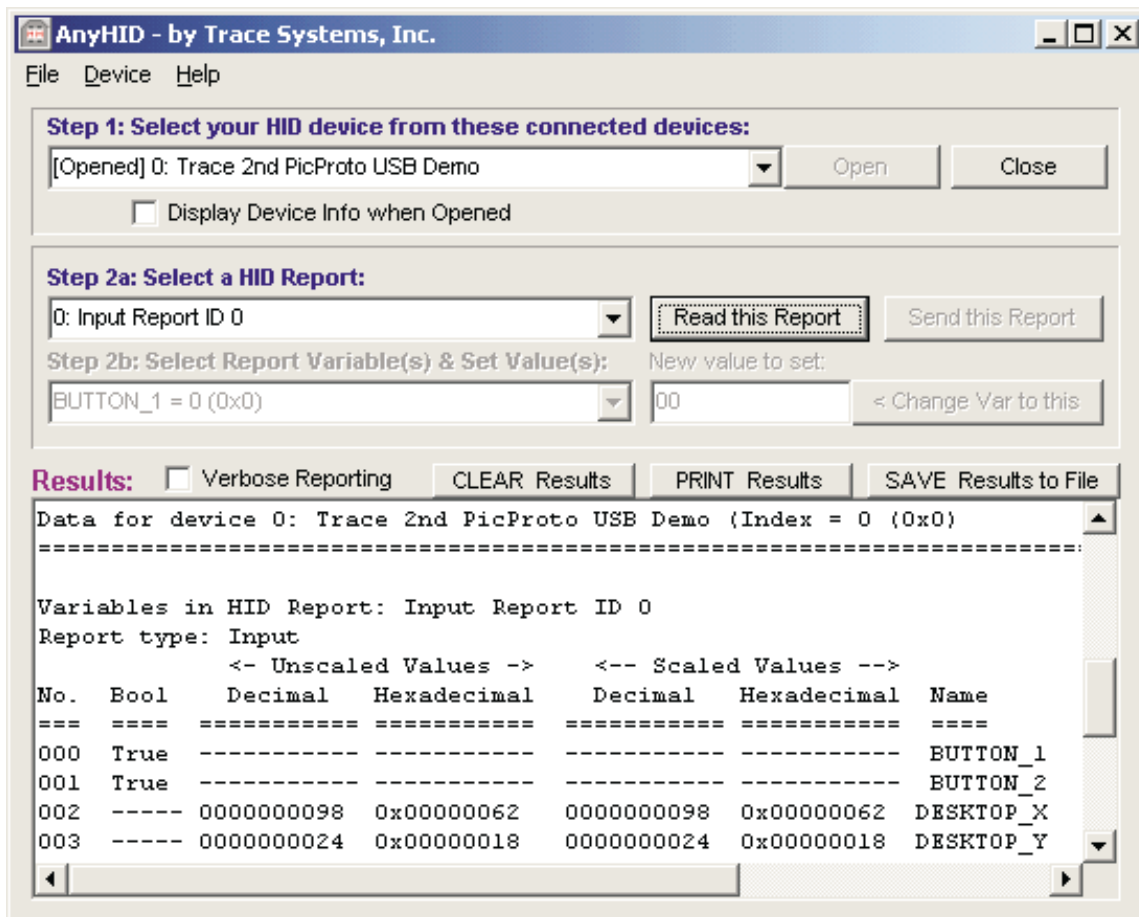
An object which represents your device. Since this object varies from one HIDmaker project to the next, the class name given to this is unique for each project. This object contains properties (of type THidVar) to represent all the HID variables. Public methods of the device object include methods to send or receive individual data reports, or to send or receive all the HID reports that are defined for the device.

The HIDmaker Software Framework makes the code very simple, clean, and exceptionally easy to understand and use. For example, to set a value to a variable, and then pack that variable into a report and send it to the device, takes only two lines of code. The Visual Basic version looks like this:

```
MyDevice.MyVar.UnscaledValue = 5
Call MyDevice.SendOutputReportA()
```

HIDmaker Test Suite

HIDmaker Combo consists of two related products bundled together. In addition to the HIDmaker program described above, HIDmaker Combo adds HIDmaker Test Suite, a set of test tools that save you time and thousands of dollars.



One part of HIDmaker is a test program called AnyHID, because it can test any properly designed USB HID device, even if you didn't design it yourself! AnyHID lets you open multiple devices or device "Interfaces," send data to and from any of them once opened, and display information about the device or individual variables.

AnyHID even lets you "browse" through all the USB HID devices connected to your PC, and learn about what's inside them before you decide to open them for I/O! If you like, you can have HIDmaker automatically open one or all USB HID devices that match your desired criteria: any combination of things like Vendor ID or Vendor Name, Product ID or Product Name, and so on.

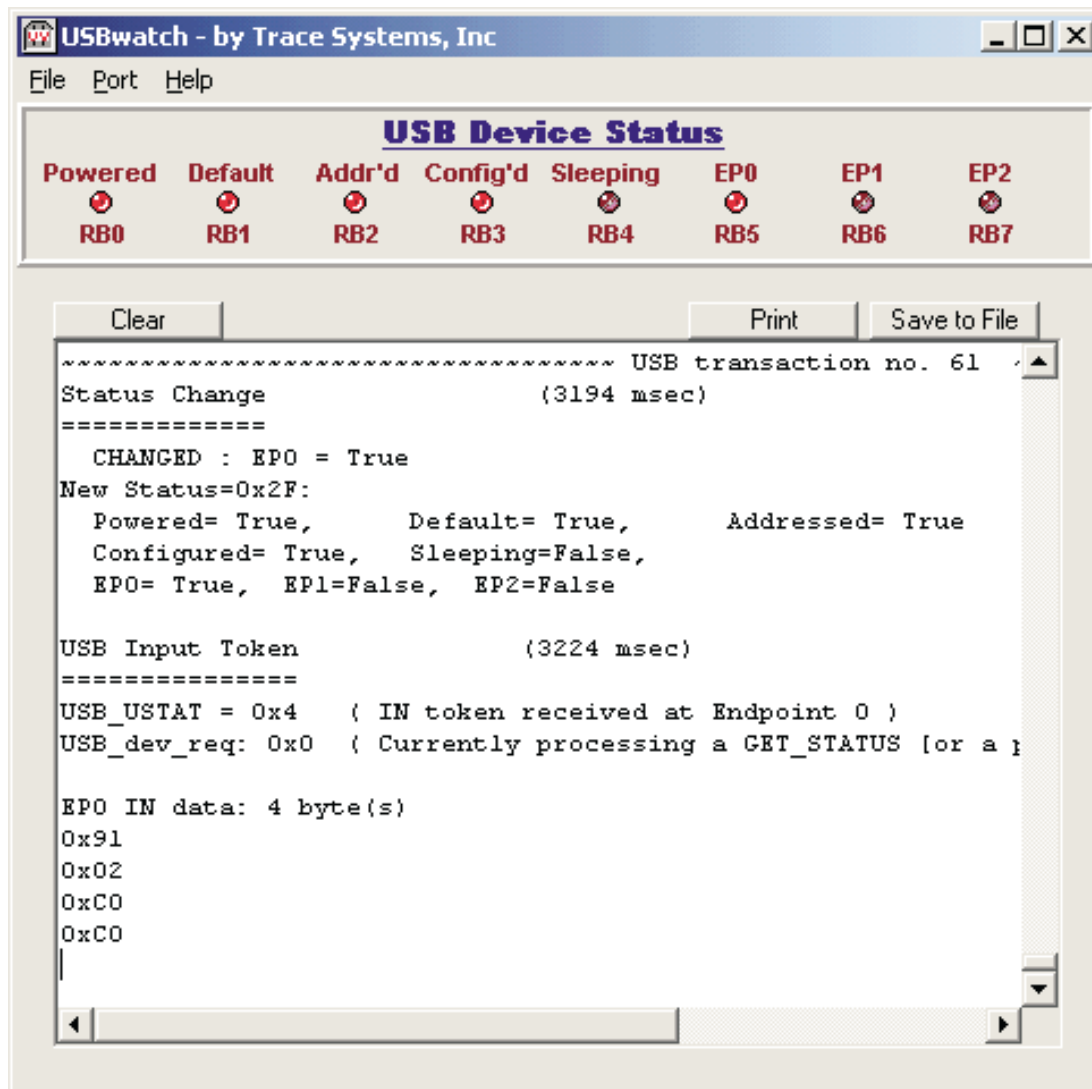
You can set the amount of detail you want AnyHID to report, and even type in your own comments to document your work. When you are ready, you can print the data, save it to a file, or copy it to the clipboard for pasting into your favorite word processor. You can even have AnyHID automatically log all data directly to a text file.

Many of these features come from the HIDagent object that is built in to all programs that are generated by HIDmaker. This means that features like browsing for HID devices, searching for matching devices, logging diagnostic data to a file, and many others are available to your other programs as well. To help our users get the most out of the HIDmaker Software Framework, we provide the source code for AnyHID, in multiple programming languages, as part of the HIDmaker Test Suite.

Another key part of the HIDmaker Test Suite is a program called USBwatch. Many people who have struggled to create their own USB peripherals without a tool like HIDmaker have learned that, if they make even one mistake in one of the complicated descriptor tables, the PC will ignore their device, and not say why. Regular USB test programs, such as the compliance test programs from the USB Implementers Forum, cannot even begin to work until you get your device working well enough for the PC to successfully "enumerate" it. If your device does not pass enumeration, you have a serious testing problem. What do you do now?

Up to now, your main option was to spend thousands of dollars on expensive USB bus analyzer tools, so you could see how far the enumeration process went before the PC gave up. Now, there's a low cost alternative: USBwatch.

USBwatch shows you the USB traffic that goes into and out of your specific device. It also shows key changes in the state of the USB engine in your device. For many of us, USBwatch provides our first chance to see what actually goes on when we send data to and from our device, and especially what Windows does when it enumerates your device. (What you find out may surprise you!)



USBwatch works by using the hardware serial port on the peripheral microcontroller to send out a few code bytes to a special USBwatch program running on a PC which has an RS-232 serial port. The USBwatch program receives this data, parses it, and displays the information. If your device fails enumeration, you can see what went wrong by comparing your test data to that taken on known good devices. (We provide examples of tests made on working devices.)

If you use HIDmaker to generate code for the microcontroller for your USB peripheral, you should never need USBwatch. However, if you modify HIDmaker's generated code, particularly the USB descriptors, you might find yourself in a situation where you have made some mistake which causes the PC to reject your device. The PC won't tell you why, but USBwatch can help you find the answer when all other approaches fail. That's why we say that USBwatch is your first line of defense against USB problems!

“Do I REALLY Need HIDmaker Combo?”

Absolutely! Nobody writes computer programs in binary anymore. We all use compilers to make our jobs easier. Writing USB programs is the same thing, and needs the same sort of tools to make things easier to understand, faster to create and debug, and more likely to actually work.

HIDmaker Software Framework helps you to think and work at a higher level. Since you don't have to spend time connecting up all the USB plumbing, you can spend your time productively, working on what matters most to you, your clients and customers: making the actual application that you set out to make in the first place.

There are a few more things we still need to talk about. One of these is the fact that the PC world is moving to USB in a big way. Already, some PCs and laptops are being sold with USB ports only: no serial or parallel ports at all. Before long, you may have trouble finding serial and parallel ports on any new computers. You need to get familiar with USB, and you need to do it SOON. HIDmaker Combo is here to help.

The cost of hardware debugging tools for USB is still very high. That's why we came up with USBwatch, but look at it this way: creating your project with HIDmaker can eliminate your need to use an expensive hardware debugging tool at all. It's true that our HIDmaker Test Suite can get you out of trouble, inexpensively, but using HIDmaker can keep you out of trouble in the first place.

HIDmaker Combo costs about as much as your favorite PC compiler. Considering how much time it will save you on every USB project, whether you're just getting started or are already a USB expert, HIDmaker Combo is a tremendous bargain.

Find out more about Dr. Bob's USB Toolbox at our web site:

http://www.tracesystemsinc.usb_tools.ivnu