Display a snapshot of Proton variables on a PC.

Quick start

* Copy SnapShot.exe and SnapShot.ini to your source directory
* Confirm that the location PPI files, comm. port and baud rate in SnapShot.ini are correct
* Set the baud rate in your code (or accept default) and include it into the program under test
* Add Dump and DumpW statements in appropriate locations to the source under test
* Compile and burn the PIC
* Connect the PIC to a PC COM port and run SnapShot.exe

"Real" serial ports are preferable, USB CDC introduces considerable latencies. See FTDI options.  
  
Getting started  
Perhaps the easiest way is to burn the demo program to one of the devices listed in SnapShotDemo.bas (see Adding a new Device if you don’t have one of these) and have a play. Extract the zip into any directory, connect up comms, try left and right clicking on columns, rows and individual items. The demo is configured to run on COM1 and uses hardware serial port on the PIC to connect to the PC, baud rate is 115200 but you can set to any standard baud rate or 0 for auto seek. If there isn’t an available hardware serial port on your test hardware, you can use any available gpio or the ICSP pins to bit bang at 9600.  
  
What it is Not  
It is not a debugger. Both P-ICD and hardware ICE offer an infinitely better debugging environment. But P-ICD is overkill for simple stuff and ICE requires you to run MPLAB or MPLABX.  
  
Motivation  
While inserting numerous "PRINT name type" into the source of my current masterpiece it occurred to me that this really wasn't optimal. What do you do when space on the LCD display was too limited for the variables of interest or where no display was available or I wanted to see the contents of an array?  
  
I wanted something that would save me the tedium of adding PRINT statements listing the variables and their type that I was interested in. I wanted something really simple, that placed few demands on the PIC's resources and would run on all PIC's with whatever external or internal peripherals. I have so far failed to achieve that last goal, in that the code uses a disproportionately large amount of ROM and RAM on most 16F’s.

Operation  
The PIC under test sends the contents of a block of memory to a PC application. The PC application parses the .LST and .BAS files of the program under test and interprets the block appropriately. To ensure that the program in the PIC and the parsed source are in "sync" the PC app is executed from the current source directory. The app reinitializes itself if the selected source is recompiled. Initially the app will look for .BAS/.LST/.HEX files in the current directory and if more than one set is found you are required to select, otherwise the app starts immediately.

There are two options for the dump statements you insert into the program under test - either dump and continue or dump and wait for the Continue button to be pressed.   
  
You have the option of displaying these additional dialogs -

* Watch - selected variables in a "stripped down" display. The important controls are duplicated so that the main display can be minimized.
* EEPROM - contents display
* Port - TRIS and data
* Source - the user defined lines of source before and after the dump point.
* Pinout - of the device under test.
* Datasheet - of the device under test
* Errata - of the device under test

On close, all dialogs save their current size, position and column widths. Watch saves its content by name not position.

Features

* Edit any variable or array subscript (18F only)
* Reset the PIC under test (18F only)
* Highlight. Red – pending edit Yellow – value changed from previous snapshot
* Auto baud rate detect – set Baud in SnapShot.ini to 0
* If used in conjunction with HexTag, firmware compilation details are displayed

Multi-shot modes

* Continuous – execution continues until deselected or another multi-shot mode is selected.
* Repeat – execution continues until a specified DumpW each time continue is pressed.
* Until – execution continues until a specified DumpW (with optional iteration count) and returns to single-shot mode

Design goals

* Should work on all Proton supported PIC's – failed, only partial support for 16F.
* Small ROM, RAM and performance footprint – failed, minimal code/compiler overhead for 16F’s is 700 program words and 50 bytes of RAM (20% of available on 16F690).
* Open source and written in Proton basic – almost, assembler equivalent of PTR8 for 16F.
* Time spent executing the include code be as short as possible – much better than expected.

Caveats  
These are numerous but principally you must be aware that a variable that appears to be unchanged between snapshots may have changed value many times but have returned to their initial value at the time of the next snapshot.

Don’t rename source files after compilation because source file names are taken from the LST file.

A whole bunch of other stuff that I’ve conveniently forgotten.

Editing

Some little explanation of the protocol is necessary to understand the limitations of editing. First up, the data displayed is as received from the PIC, no assumptions are made. Therefore edited values are displayed but highlighted in red, until the next time the PIC sends a transaction. When not sending a transaction or running code under test, the PIC will either be requesting permission to send data or requesting permission to run code under test. Thus, the two windows for sending edits, either before or after dumping data. Flags are available to enable either or both of these modes, each with its own limitations.

Customization  
You can freely (well almost) modify the code in the include file. You could perhaps set a wait condition dependent on some variable or on a local counter. the following example might be inserted after the label D\_Dump  
  
  IF SomeVarable = SomeValue THEN  
      D\_WaitFlag = True  
  ELSE  
      D\_WaitFlag = False

  ENDIF

FTDI options

If you are using FDTI drivers, in Port settings/Advanced under BM Options, set Latency Timer to it’s minimum value.

Adding a new Device

* Add its signature to the SnapShot.ini file (optional if device is in PicKit2 database)
* In the Device Specific section of SnapShot.inc
  + add specifics of PGD and PGC
  + add address of Device ID
  + add the specifics to achieve the various baud rates
  + add any clock config/initialisation code