PICkit 2 Interface Guide

PC Application v2.52

OS Firmware v2.32

Document #:	
Title:	**PICkit 2 Interface Guide**
Subtitle:	Matches Firmware Version 2.32.00, and software V2.51.00
Date:	July 28, 2008
Description:	This document presents information on the design of the PICkit 2 PC application and OS firmware, and the interfaces between the Device File, Application, and Firmware.

1.0 Introduction

This document covers design and theory of the PICkit 2 version 2 firmware and PC application software. Version 2 implements a script-based design to allow maximum flexibility for programming any type of device without requiring frequent, part-specific changes and updates. All part-specific information is kept in a device file, allowing new parts and families with new algorithms to be implemented without frequent changes to the software or firmware.

2.0 Guiding Design Goals

The guiding design goals of the version 2 firmware

1. Allow the firmware and software to be as generic as possible so adding new part & family support requires little to no changes.
2. Remain compatible with bootloader commands for the existing Pk2.
3. All device specific information should be kept in the device file.
4. The scripting and firmware should be flexible enough to allow other tasks besides programming to be accomplished, such as ICD.

3.0 Architecture Overview

The PICkit 2 is organized into 4 functional blocks.

1. Bootloader
2. Programmer Firmware
3. PC Application
4. Device File

The Bootloader is designed to be autonomous i.e. does not require programmer firmware for operation.
The Programmer firmware, however, has its own USB stack, and does not rely on the bootloader for communication while running. This allows the USB stack used during normal execution to be updated over USB using the bootloader.
The programmer firmware is largely a script execution engine, with basic commands for management of scripts and data. Data is program code, EE contents, and other information for which the programmer only acts as a conduit.
The PC application is responsible for the user interface portion of the system. Its function is to respond to user requests by packaging data and scripts for execution by the firmware.
The device file contains all scripts and descriptions of each part supported. It is divided into 3 main sections. The first is the family description section, which sets up some parameters common to all family members. The second is the part description section, with an entry for each part supported. This description references scripts contained in the third section, and defines membership in a family defined in section 1. The third section contains an indexed listing of all scripts used by the supported parts.

4.0 Bootloader

- TBD - .

5.0 Programmer Firmware

The firmware consists largely of commands to store and retrieve data from 2 data buffers, and to store and execute scripts from an indexable script buffer.
The names & uses of the buffers are:

Download Data Buffer
256 byte circular buffer. Stores data sent by the PC host for use by script commands. This is where code and data to be programmed by a script is stored. The write pointer is advanced by USB commands that write to the buffer, and the read pointer is advanced by scripts that use the data.

Upload Data Buffer
128 byte circular buffer. Data retrieved by a script is stored here to be read over USB by the PC host. Data and code read from a part by a script is stored here. The write pointer is advanced by scripts that place data into the buffer, and the read pointer is advanced by USB commands to retrieve the data.

Script Buffer
768 byte indexed buffer. This buffer stores up to 32 scripts of variable sizes up to 768 bytes total. The Script Index Table stores the beginning location and length of each script 1 through 32. Currently scripts are limited in length to 62 bytes as that is the longest script than can be sent via a DOWNLOAD_SCRIPT command in a 64-byte USB packet.

5.1 Firmware Commands

A command packet starts with the command byte and is followed by the command data bytes in order starting with data[1]. Response data bytes are returned via USB starting at data[1].
Any command not defined is treated as a “No Operation”.

Byte ID	Command Name		Data Length	Data Description
0x42	**ENTER_BOOTLOADER**	command:	0	Transfers operation to bootloader.
		response:	0	
0x5A	**NO_OPERATION**	command:	0	Command processor skips and goes to next command byte.
		response:	0	
0x76	**FIRMWARE_VERSION** - Read firmware version	command:	0	

		response:	3	data[1] = Major version
				data[2] = Minor version
				data[3] = Dot version
				
				When received as the first byte in a USB packet, exits PK2GO mode.

0xA0	**SETVDD** - Set Target VDD voltage VDDLim is error detection threshold voltage.	command:	3	data[1] = CCPL data[2] = CCPH = [((Vdd * 32) + 10.5) << 6] data[3] = VDDLim = (Vfault / 5) * 255 typ. Vfault = 0.7*Vdd data[3] is calibrated using the factors sent via **SET_VOLTAGE_CALS** (0xB0)
		response:	0	

0xA1	**SETVPP** - Set Target VPP voltage VPPLim is error detection threshold voltage.	command:	3	data[1] = CCPR2L = 0x40 data[2] = VPPADC = [Vpp * 18.61] data[3] = VPPLim = [Vfault * 18.61] typ. Vfault = 0.7*Vpp data[2] & data[3] are calibrated using the factors in **SET_VOLTAGE_CALS** (0xB0)
		response:	0	
0xA2	**READ_STATUS** - Read status word. StatusHigh and StatusLow 7:4 are cleared after a read.	command:	0	
		response:	2	data[1] = Status Low <7> unused <6> 1 = PICkit 2 button pressed <5> 1 = VppError (Vpp < Vfault) <4> 1 = VddError (Vdd < Vfault) <3> 1 = Vpp On <2> 1 = Vpp GND On <1> 1 = Vdd On <0> 1 = Vdd GND On data[2] = Status High <7> 1 = Download Buffer Overflow <6> 1 = Script Buffer Overflow, invalid Script length or index <5> 1 = Run Script on Empty Script <4> 1 = Script abort- download empty <3> 1 = Script abort- upload full <2> 1 = ICD transfer timeout/Bus Error <1> 1 = UART Mode enabled <0> 1 = PICkit 2 reset since last Status Read. NOTE: Turns BUSY_LED off.
0xA3	**READ_VOLTAGES** - Read Vdd & Vpp voltages	command:	0	

		response:	4	data[1] = VDDADCLow
				data[2] = VDDADCHigh
				Vdd = (VDDADC / 65536) * 5 V
				data[3] = VPPADCLow
				data[4] = VPPADCHigh
				Vpp = (VDDADC / 65536) * 13.7 V
				
				VDDADC and VPPADC value are calibrated using the values in **SET_VOLTAGE_CALS** (0xB0)

0xA4	**DOWNLOAD_SCRIPT**
	- Stores a script in the Script Buffer. Any existing script at The script# will be replaced.
		command:	2+N	data[1] = Script # (0-31)
				data[2] = Script Length N
				data[3] = Script byte 1
				xxx
				data[N+2] = Script byte N

		response:	0	

0xA5	**RUN_SCRIPT***
	- Runs a script from the Script Buffer.
		command:	2	data[1] = Script # (0-31)
				data[2] = Run 1 – 256 times
				(0x00 = 256x, 0x01 = 1x, 0xFF = 255x)

		response:	0	

0xA6	**EXECUTE_SCRIPT*** - Executes the included script.	command:	1+N	data[1] = Script Length N
				data[2] = Script byte 1
				xxx
				data[N+1] = Script byte N

		response:	0	
0xA7	**CLR_DOWNLOAD_BUFFER** - Empties the Download Buffer	command:	0	
		response:	0	

0xA8	**DOWNLOAD_DATA** - Adds data to end of buffer	command:	1+N	data[1] = Data Length N
				data[2] = Data byte 1
				xxx
				data[N+1] = Data byte N

		response:	0	
0xA9	**CLR_UPLOAD_BUFFER** - Empties the Upload Buffer	command:	0	
		response:	0	
0xAA	**UPLOAD_DATA** - Read data from Upload Buffer	command:	0	

		response:	1+N	data[1] = Data Length N
				data[2] = Data byte 1
				xxx
				data[N+1] = Data byte N

0xAB	**CLR_SCRIPT_BUFFER** - Empties the Script Buffer	command:	0	
		response:	0	
0xAC	**UPLOAD_DATA_NOLEN** - Read data from Upload Buffer, no preceding length byte.	command:	0	

		response:	N	data[1] = Data byte 1
				xxx
				data[N] = Data byte N

0xAD	**END_OF_BUFFER**	command:	0	Indicates end of commands in USB 64-byte report buffer.
		response:	0	
0xAE	**RESET** - Resets the PICkit 2	command:	0	Any other commands in the buffer after the RESET will not be executed.
		response:	0	
0xAF	**SCRIPT_BUFFER_CHKSM** - Read checksums of the script buffer	command:	0	

		response:	4	data[1] = LengthSumL
				data[2] = LengthSumH
				- 16-bit sum of the lengths of all loaded scripts
				data[3] = BufferSumL
				data[4] = BufferSumH
				- 16-bit sum of all used bytes in script buffer

0xB0	**SET_VOLTAGE_CALS**
	- Sets the calibration factors for VDD setpoints and ADC conversions.
	- Stored in internal EE and used on subsequent resets.
		command:	4	data[1] = adc_calfactorL
				data[2] = adc_calfactorH
				CalibratedResult =
				(ADRES * adc_calfactor) >> 8
				data[3] = vdd_offset (signed 2's comp)
				data[4] = vdd_calfactor
				Calibrated CCP value =
				(((CCP >> 6) + vdd_offset) * vdd_calfactor) >>1
				
				Default (uncalibrated) values:
				data[1] = 0x00, data[2] = 0x01, data[3] = 0x00, data[4] = 0x80,

		response:	0	

0xB1	**WR_INTERNAL_EE**
	- Writes data to internal EEPROM
		command:	2 + N	data[1] = Start Address
				data[2] = Data Length N (Max N=32)
				data[3] = Data byte 1
				xxx
				data[N+2] = Data byte N
				
				First data byte written at Start Address. Additional bytes are written at subsequent addresses.

		response:	0	

0xB2	**RD_INTERNAL_EE**
	- Read data from internal EEPROM
		command:	1	data[1] = Start Address
				data[2] = Data Length N (Max N=32)
				
				First data byte read from Start Address. Additional bytes are read at subsequent addresses.

		response:	N	data[1] = Data byte 1
				xxx
				data[N] = Data byte N

0xB3	**ENTER_UART_MODE**
	- Puts the PICkit 2 into UART Mode.
	- Scripts will not execute in UART Mode.
		command:	2	data[1] = BaudValueL
				data[2] = BaudValueH
				
				BaudValue = 65536 – [((1/BAUD) – 3e-6) / 1.67e-7]
				Where BAUD = 1200, 2400, etc to maximum 57600.

		response:	0	
0xB4	**EXIT_UART_MODE** - Returns PICkit 2 to normal mode.	command:	0	
		response:	0	

0xB5	**ENTER_LEARN_MODE**
	- Puts PICkit 2 into "Learn" mode, where commands, scripts, and data are stored in the external EEPROM.
		command:	4	data[1] = 0x50
				data[2] = 0x4B
				data[3] = 0x32
				data[4] = 0 : 128K EEPROM
				1 : 256K EEPROM
				
				First 3 bytes are command "key" value to prevent accidental mode entry.

		response:	0	

0xB6	**EXIT_LEARN_MODE** - Returns PICkit 2 to normal mode.	command:	0	
		response:	0	

0xB7	**ENABLE_PK2GO_MODE**
	- Returns PICkit 2 to normal mode.
		command:	2	data[1] = 0x50
				data[2] = 0x4B
				data[3] = 0x32
				data[4] = 0 : 128K EEPROM
				1 : 256K EEPROM
				
				First 3 bytes are command "key" value to prevent accidental mode entry. 4th byte tells what EEPROMs to use (2 x 24LC512 vs 2 x 24LC1025).
				PK2GO Mode is exited by the reception of a FIRMWARE_VERSION command in the first byte of a USB packet.

		response:	0	
0xB8	**LOGIC_ANALYZER_GO** - Logic Analyzer function. - NOTE: clears script buffer	command:	7	data[1] = 0 : Falling Edge Triggers 1 : Rising Edge Triggers data[2] = TrigMask – '1' bit for active Ch data[3] = TrigStates – Edge bits 2nd state data[4] = EdgeMask – '1' bit for edge Ch data[5] = TrigCount '0' = 256 counts data[6] = PostTrigCountL data[7] = PostTrigCountH data[8] = SampleRateFactor 0 : 1 MHz 1 : 500 kHz 3 : 250 kHz 9 : 100 kHz 19 : 50 kHz 39 : 25 kHz 99 : 10 kHz 199 : 5 kHz Mask/State bits: Ch1 = bit 2 Ch2 = bit 3 Ch3 = bit 4 Unused Mask/State bits should be '0' PostTrigCount is the number of extra samples to be taken after the trigger, minus 1. Min value is 1. Max is 65279
		response:	2	data[1] = TrigLocL data[2] = TrigLocH bit 7 = 1 : trigger is in swapped sample bit 6 = 1 : trigger is an abort !! Add 1 to get the actual address !! TrigLoc is an address 0x600 to 0x7FF. On an abort, the address is invalid

0xB9	**COPY_RAM_UPLOAD**
	- Copies 128 bytes of RAM to the Upload buffer starting at the given address.
		command:	2	data[1] = StartAddressL
				data[2] = StartAddressH
				
				bits 15-12 of StartAddress are ignored.

		response:	0	
LEARNING MODE META-COMMANDS These commands are only valid in Learning Mode.				

0x80	READ_OSCCAL	Command:	2	data[1] = OSCCAL Address Low
				data[2] = OSCCAL Address High
				
				Reads & stores the device OSCCAL value

0x81	WRITE_OSCCAL	Command:	2	data[1] = OSCCAL Address Low
				data[2] = OSCCAL Address High
				
				Writes the device OSCCAL value stored by READ_OSCCAL

0x82	START_CHECKSUM	Command:	2	data[1] = Format
				data[2] = 0
				
				Format = 0 for most devices
				= 1 for Baseline / Midrange flash
				= 2 for Baseline / Midrange EE
				
				Begin calculating a checksum on the upload buffer.

0x83	VERIFY_CHECKSUM	Command:	2	data[1] = Checksum Low
				data[2] = Checksum High
				
				Compares calculated checksum against included checksum.

0x84	CHECK_DEVICE_ID	Command:	2	data[1] = DeviceIDMaskL
				data[2] = DeviceIDMaskH
				data[3] = DeviceIDValueL
				data[4] = DeviceIDValueH
				
				Checks that the target Device ID matches the argument value.

0x85	READ_BANDGAP	Command:	0	Reads the bandgap of 12F629 family devices
0x86	WRITE_CFG_BANDGAP	Command:	0	Writes the Config word of 12F629 family devices with the read badgap value.

0x87	CHANGE_CHKSM_FRMT	Command:	2	data[1] = Format
				data[2] = 0
				
				Format = 0 for most devices
				= 1 for Baseline / Midrange flash
				= 2 for Baseline / Midrange EE

* If a script attempts to use data from the download buffer and it is empty, the script will abort and generate a status error. If a script attempts to place data in the upload buffer and it is full, the script will abort and generate a status error.

5.1.1 Command USB Responses

Each command that provides a response will generate a separate USB read packet with only the data from that command. Each packet is set up as a blocking USB read. Thus, multiple commands may be stacked into a single USB Write packet, but a seperate USB read must be completed to retrieve the result for each command that returns a response. The response packets will be read in the order in which the commands appear in the Write packet. Results from multiple commands in a single Write packet are not appended to a single Read packet.
This allows data to immediately available from a command after it has executed, without waiting for the entire USB Write packet to be processed. Data can be read as it is available even if a script with a long execution time (such as bulk erase) appears later in the command list.

5.1.2 UART Mode

UART Mode allows the PICkit 2 to be used as a simple UART.
ICSPCLK = TX (data transmitted from the PICkit 2)
ICSPDAT = RX (data received from target UART)

Signal levels are inverted (ie Start Bit = GND) logic level signals between GND and VDD.

A bit in the high status byte returned by the READ_STATUS command allows determination if the PICkit 2 is in UART Mode or not.

> Restrictions:
> VPP will be shut off when UART mode is entered.
>
> Scripts will not be run in UART mode if script commands are received.
>
> PICkit 2 should NOT supply VDD in UART mode, as the Voltage Error Detection interrupt service routines are switched off as they would interfere with the UART interrupt service routines. Thus, supplying VDD in UART mode carries the danger of a VDD short causing damage to the PICkit 2 unit or USB host PC.
> The PICkit 2 VDD pin, however, does need to connected to the target VDD so the ICSP signals are clamped to the proper voltage.

In UART mode, any data written to the Download Data Buffer will be transmitted on the TX pin at the specified baud rate.

Any received data on the RX pin will be placed in the Upload Data Buffer. The PC host should poll the buffer for data using the UPLOAD_DATA command. The Length byte in the response packet allows determination if any data was received.
Note that the Upload Data Buffer may be overrun if the host does not keep up with the received data rate. Overflow data will be lost.

Data Buffers should be cleared before putting the unit in UART Mode

The BUSY LED acts as an activity light (for both RX and TX).

Baud rates faster than 38400 may not work reliably.

5.1.3 Unit IDs

A PICkit 2 unit may be assigned a unique Unit ID string of up to 15 characters. This string is stored in 16 bytes of internal EEPROM on the PICkit 2 PIC18F2550. A value string is always preceded by the char byte '#' to indicate a valid string. If this first byte is not '#', the Unit ID is assumed to be blank.

Starting with PICkit 2 Firmware v2.32.00, the Unit ID is returned in the PICkit 2USB Descriptor for Serial Number String. This allows the Unit ID to be read from the USB descriptors without sending any HID commands, so software may detect and list all attached PICkit 2 units without risking corrupting any current communications with a PICkit 2 unit if it were to send HID commands.

5.2 Scripting

Execution of a script will abort on the following conditions:

- Script is attempting to use bytes from the Download Data Buffer and the buffer is empty.
- Script is attempting to store bytes in the Upload Data Buffer, and the buffer is full.
- ICD Timeout or BusError bit is set.

RUN_SCRIPT and EXECUTE_SCRIPT commands will be ignored when any of StatusHigh bits 7:1 are set (i.e. a script error exists) until the error is acknowledge and cleared by a READ_STATUS command.

Some script control bytes require arguments. Argument bytes follow the control byte directly, in the order given in the table. All undefined byte values will be interpreted as "no operation."

ID	Control Byte	Arguments	Description.
0xFF	VDD_ON	-	Turns on VDD PFET pass transistor.
0xFE	VDD_OFF	-	Turns off VDD PFET pass transistor.
0xFD	VDD_GND_ON	-	Turns on VDD NFET ground.
0xFC	VDD_GND_OFF	-	Turns off VDD NFET ground.
0xFB	VPP_ON	-	Turns on VPP PNP pass transistor.
0xFA	VPP_OFF	-	Turns off VPP PNP pass transistor.
0xF9	VPP_PWM_ON	-	Start up VPP PWM. Allow 100ms to come up to voltage.
0xF8	VPP_PWM_OFF	-	Shut down VPP PWM. Q4 is off (VPP_PUMP low).
0xF7	MCLR_GND_ON	-	Turns on VPP NPN ground.
0xF6	MCLR_GND_OFF	-	Turns off VPP NPN ground.
0xF5	BUSY_LED_ON	-	Turns on BUSY LED
0xF4	BUSY_LED_OFF	-	Turns off BUSY LED
0xF3	SET_ICSP_PINS	arg[1] = pin states <7 – 4> unused <3> PGD logic level <2> PGC logic level <1> 1= PGD input, 0= output <0> 1= PGC input, 0= output	Sets the logic level and direction of the ICSP pins. The logic level is always set first, before the direction takes effect.
0xF2	WRITE_BYTE_LITERAL	arg[1] = Byte to be sent	Clocks out the following byte on PGC, PGD. Byte is shifted LSB first. PGD and PGC must be set to outputs before calling.
0xF1	WRITE_BYTE_BUFFER	-	Clocks out the next byte from the Downstream Data Buffer and advances the read pointer. Byte is shifted LSB first. ** PGD and PGC must be set to outputs before calling.
0xF0	READ_BYTE_BUFFER	-	Clocks in a byte of data and stores it in the Upstream Data Buffer, advancing the write pointer. Byte is shifted in LSB first. ** PGC must be set to output. PGD is automatically set an input, and restored to previous state after read.

0xEF	READ_BYTE	-	Clocks a byte of data, but throws it away (does not place in buffer). PGC must be set to output. PGD is automatically set an input, and restored to previous state after read.

0xEE	WRITE_BITS_LITERAL	arg[1] = N bits (1-8)
		arg[2] = literal
			Clocks out the first N LSbits of 'literal'
			PGD and PGC must be set to outputs before calling.
			Set PGD, delay, PGC=1,delay, PGC=0

0xED	WRITE_BITS_BUFFER	arg[1] = N bits (1-8)	Clocks out the first N LSbits of the next byte in the Downstream Data Buffer and increments the read pointer. PGD and PGC must be set to outputs before calling. Set PGD, delay, PGC=1,delay, PGC=0
0xEC	READ_BITS_BUFFER	arg[1] = N bits (1-8)	Clocks in the first N LSbits into the next byte in the Upstream Data Buffer and increments the write pointer. PGC must be set to output. PGD is automatically set an input, and restored to previous state after read.
0xEB	READ_BITS	arg[1] = N bits (1-8)	Clocks in N bits, but throws away the results. PGC must be set to output. PGD is automatically set an input, and restored to previous state after read.
0xEA	SET_ICSP_SPEED	arg[1] = rate	Rate byte gives the clock period in multiples of 1us.
0xE9	LOOP	arg[1] = Index offset to loop to arg[2] = Loop iterations	The *index offset* is unsigned (always backwards), and is the number of bytes back from the 0xE9 control byte to loop. Value '1' will loop to the byte prior to the 0xE9. *Loop iterations* gives the number of times the script loop is executed again. '0'=256x NOTE: LOOP commands may NOT be nested, but may be nested with a LOOPBUFFER command.
0xE8	DELAY_LONG	arg[1] = Length of delay	Each unit of delay is 5.46ms. '0'= 256 units.
0xE7	DELAY_SHORT	arg[1] = Length of delay	Each unit of delay is 21.3us. '0'= 256 units.
0xE6	IF_EQ_GOTO*	arg[1] = Byte for comparison arg[2] = Index offset for goto.	If the last byte stored in the Upload buffer is equal to arg[1], then script execution will branch to the offset from the GOTO control byte given by arg[2]. The branch offset is signed 2's complement.
0xE5	IF_GT_GOTO*	arg[1] = Byte for comparison arg[2] = Index offset for goto.	If the last byte stored in the Upload buffer is greater than arg[1], then script execution will branch to the offset from the GOTO control byte given by arg[2]. The branch offset is signed 2's complement. The compare is unsigned.

0xE4	GOTO_INDEX*	arg[1] = Index offset for goto	Branch to the given offset from the GOTO control byte. The branch offset is signed 2's complement.
0xE3	EXIT_SCRIPT	-	The script stops executing immediately. Normally, execution ends when the end of the script is reached.
0xE2	PEEK_SFR	arg[1] = 8 LSbs of SFR address	The contents of the given SFR are placed in the Upload Buffer
0xE1	POKE_SFR	arg[1] = 8 LSbs of SFR address arg[2] = byte to write to SFR	Writes the given literal into the given SFR.
0xE0	ICDSLAVE_RX	-	Handles the ICD handshake and receives a byte of data from the DE, which is placed in the Upload Buffer.
0xDF	ICDSLAVE_TX_LIT***	arg[1] = byte to be transmitted	Handles the ICD handshake and allows the argument byte to be clocked out by the DE.
0xDE	ICDSLAVE_TX_BUF***	-	Handles the ICD handshake and allows the next byte in the download buffer to be clocked out by the DE.
0xDD	LOOPBUFFER	arg[1] = Index offset to loop to	The *index offset* is unsigned (always backwards), and is the number of bytes back from the 0xDD control byte to loop. Value '1' will loop to the byte prior to the 0xDD. *The number of loop iterations is taken from the download buffer as a word – the LSB is first in the buffer. 0x0000 = NO iterations. This is different from the LOOP command. These bytes are read from the download buffer when the LOOPBUFFER control byte is first reached.* NOTE: LOOPBUFFER commands may NOT be nested, but may be nested with a LOOP command. – provided interation values exist in the download buffer for each LOOP iteration.
0xDC	ICSP_STATES_BUFFER	-	Puts the ICSP pin states in the Upload buffer. <7 – 2> unused = 0 <1> PGD pin, 1= high, 0=low <0> PGC pin, 1= high, 0=low
0xDB	POP_DOWNLOAD	-	Reads and discards a byte from the Download Buffer.

0xDA	COREINST18	arg[1] = lsb
		arg[2] = msb
			Shifts out 4 bits of zeroes, followed by arg1 then arg2

0xD9	COREINST24	arg[1] = low byte
		arg[2] = mid byte
		arg[3] = upper byte
			Shifts out 4 bits of zeroes, followed by arg1, arg2, then arg3.

0xD8	NOP24		Shifts out 22 bits of zeroes.
0xD7	VISI24		Shifts out 4 bits b0001, a byte of zeroes, then shifts in 2 bytes into the upload buffer.

0xD6	RD2_BYTE_BUFFER	-	Clocks in a byte of data and stores it in the Upstream Data Buffer, advancing the write pointer. Byte is shifted in LSB first. Data is latched on rising edge of clock PGC must be set to output. PGD is automatically set an input, and restored to previous state after read.
0xD5	RD2_BITS_BUFFER	arg[1] = N bits (1-8)	Clocks in the first N LSbits into the next byte in the Upstream Data Buffer and increments the write pointer. Data is latched on rising edge of clock PGC must be set to output. PGD is automatically set an input, and restored to previous state after read.
0xD4	WRITE_BUFWORD_W	arg[1] = PIC24 W register #	Forms & clocks 16-bit core instruction 0x2xxyyn : MOV #xxyy, Wn Where n= arg1 yy = first byte in download buffer xx = next byte in download buffer (includes SIX command)
0xD3	WRITE_BUFBYTE_W	arg[1] = PIC24 W register #	Forms & clocks 16-bit core instruction 0x200yyn : MOV #00yy, Wn Where n= arg1 yy = first byte in download buffer (includes SIX command)
0xD2	CONST_WRITE_DL	arg[1] = byte constant	Writes arg[1] byte to the Download Buffer.

0xD1	WRITE_BITS_LIT_HLD	arg[1] = N bits (1-8)
		arg[2] = literal
			Clocks out the first N LSbits of 'literal'
			PGD and PGC must be set to outputs before calling.
			Differs from WRITE_BITS_LITERAL in that the instead of setting data, delay, clock high, delay, clock low, this routine works as setting data, clock high, delay, clock low, delay. – for when hold time after clock falls is important.

0xD0	WRITE_BITS_BUF_HLD	arg[1] = N bits (1-8)	Clocks out the first N LSbits of the next byte in the Downstream Data Buffer and increments the read pointer. PGD and PGC must be set to outputs before calling. Differs from WRITE_BITS_BUFFER in that the instead of setting data, delay, clock high, delay, clock low, this routine works as setting data, clock high, delay, clock low, delay. – for when hold time after clock falls is important.

0xCF	SET_AUX	arg[1] = pin state <7 – 2> unused <1> AUX logic level <0> 1=AUX input, 0=output	Sets the logic level and direction of the AUX pin. The logic level is always set first, before the direction takes effect.
0xCE	AUX_STATE_BUFFER	-	Puts the AUX pin state in the Upload buffer. <7 – 1> unused = 0 <0>AUX pin, 1= high, 0=low
0xCD	I2C_START	-	Set pins prior PGC=outpt 1, AUX = inpt PGC = SCL (push-pull) AUX = SDA (open drain) Creates a START condition on the bus lines. (Sets SCL first) (Assumes SDA is high)

0xCC	I2C_STOP	-	Set pins prior PGC=output, AUX = inpt
			PGC = SCL (push-pull)
			AUX = SDA (open drain)
			Creates a STOP condition on the bus lines. (Clears SDA, SCL first)

0xCB	I2C_WR_BYTE_LIT	arg[1] = Byte to be sent	[Set pins prior PGC=output, AUX = inpt
			PGC = SCL (push-pull)
			AUX = SDA (open drain)]
			Writes byte MSb first to bus. If no ACK, sets “Bus Error” in READ_STATUS

0xCA	I2C_WR_BYTE_BUF	-	[Setup applies] Writes byte from download buffer MSb first to bus. If no ACK, sets “Bus Error” in READ_STATUS
0xC9	I2C_RD_BYTE_ACK	-	[Setup applies] Reads a byte from bus, MSb first, in to upload buffer. Generates an ACK.
0xC8	I2C_RD_BYTE_NACK	-	[Setup applies] Reads a byte from bus, MSb first, in to upload buffer. Does not generate an ACK.

0xC7	SPI_WR_BYTE_LIT	arg[1] = Byte to be sent	[Set pins prior PGC = output 0, PGD = input, AUX = output 0
			PGC = SCK
			PGD = SI
			AUX = SO
			Byte to be sent is clocked out MSb first, with MSB bit clock edge the first rising edge.

0xC6	SPI_WR_BYTE_BUF	-	[Setup applies] Byte to be sent from download buffer is clocked out MSb first, with MSb bit clock edge the first rising edge.
0xC5	SPI_RD_BYTE_BUF	-	[Setup applies] Byte is read MSb first, with first byte clocked in on first rising edge of clock.

0xC4	SPI_RDWR_BYTE_LIT	arg[1] = Byte to be sent	[Setup applies] Write byte is clocked out MSb first, with MSB bit clock edge the first rising edge. Read byte is read MSb first, with first byte clocked in on first rising edge of clock, into upload buffer.
0xC3	SPI_RDWR_BYTE_BUF	-	[Setup applies] Write byte from download buffer is clocked out MSb first, with MSB bit clock edge the first rising edge. Read byte is read MSb first, with first byte clocked in on first rising edge of clock, into upload buffer.
0xC2	ICDSLAVE_RX_BL	-	Handles the ICD handshake and receives a byte of data from the DE, which is placed in the Upload Buffer.
0xC1	ICDSLAVE_TX_LIT_BL	arg[1] = byte to be transmitted	Handles the ICD handshake and allows the argument byte to be clocked out by the DE.
0xC0	ICDSLAVE_TX_BUF_BL	-	Handles the ICD handshake and allows the next byte in the download buffer to be clocked out by the DE.
0xBF	MEASURE_PULSE	-	Measure up to 700ms positive pulse on PGD pin (must be set as input previously). Pulse must start within 700ms of measure start or it times out. Pulse length is stored in UploadBuffer as a Word in 21.333us increments. If value is zero, no pulse detected. If value is 0xFFFF pulse is longer than 700ms. Measurement offset of up to +12/-16 counts.

0xBE	UNIO_TX	arg[1] = Device Address
		arg[2] = N bytes to transmit
			Transmits N bytes from DL buffer on AUX pin, after transmitting a Start Header (with TSS) & Device Address. NoMAKs the last byte. AUX pin set to output at start, input when done. If NoSAK received on transmit byte, sets status bit ICD transfer timeout/Bus Error. Bytes sent MSb first

0xBD	UNIO_TX_RX	arg[1] = Device Address
		arg[2] = N bytes to transmit
		arg[3] = X bytes to receive
			Transmits N bytes from DL buffer on AUX pin, after transmitting a Start Header (with TSS) & Device Address. AUX pin set to output at atart, input when done. Then receives X bytes into UL buffer, NoMaks the last byte. If NoSAK received, sets status bit ICD transfer timeout/Bus Error. Bytes received MSb first

0xBC	JT2_SETMODE	arg[1] = number of bits
		arg[2] = TMS value
			JTAG 2-Wire SetMode([1]'[2])
			Bits in arg[2] sent LSb first. TDI = 0.

0xBB	JT2_SENDCMD	arg[1] = Command Value	Executes JTAG 2-Wire Send Comand with 5 bit command in 5 LSbs of arg[1]
0xBA	JT2_XFERDATA8_LIT	arg[1] = byte value to transfer	Uses the XferData pseudo op to send 8 bits of data. Received data is placed in the Upload Buffer.

0xB9	JT2_XFERDATA32_LIT	arg[1] = LSB of value to transfer
		arg[2] =2nd byte
		arg[3] = 3rd byte
		arg[4] = MSB
			Transmits the given 32-bit value and places the received 32-bit value in the Upload buffer, LSB first.

0xB8	JT2_XFRFASTDAT_LIT	arg[1] = LSB of value to transfer
		arg[2] =2nd byte
		arg[3] = 3rd byte
		arg[4] = MSB
			Transmits the given 32-bit literal value. If PrAcc= 0, sets Status -> BusError

0xB7	JT2_XFRFASTDAT_BUF	-	Transmits a 32-bit value from the Download Buffer. If PrAcc= 0, sets Status -> BusError
0xB6	JT2_XFERINST_BUF	-	Completes the XferInstruction pseudo op. 32-bit <instruction> is pulled from the Download Buffer.
0xB5	JT2_GET_PE_RESP	-	Completes the GET PE RESPONSE pseudo op and places the 32-bit value in the Upload Buffer.
0xB4	JT2_WAIT_PE_RESP	-	Completes the GET PE RESPONSE pseudo op but dumps the received data.
0xB3	JT2_PE_PROG_RESP		Completes the GET PE RESPONSE pseudo op but dumps the received data and skips the "Tell CPU to execute inst" commands.

*GOTO statements may NOT be placed within LOOPs.
** It is expected that for writes of baseline and midrange 12/14-bit instructions will be packaged by the host software into a word with start and stop bits, which will be split between 2 subsequent bytes. On reads, the host software will need to unpack the instructions from two subsequent bytes which include start and stop bits.
*** These routines leave ICSPDAT pin (RA2) as an output in the state of the last bit transmitted. This is so that very slow debug targets will not miss the last bit.

6.0 Programmer-To-Go

Target blinks twice = ready to program / last operation successful

Busy blinks quickly constantly – Vdd and/or Vpp error
Busy blinks repeating:
- 2 – DeviceID error
- 3 – Verify error.
- 4 – internal error (unrecoverable)

7.0 PC Application

The PC application loads the Device File into memory during startup. All families are loaded into an array of structures, indexed by family ID. All part parameters are loaded into a linked list of structures. All scripts loaded into an array which is indexed by script number.

NOTE: All references in following sections to "Check for connected device" include checks for voltage errors and self-powered/unpowered target devices.

7.1 Application Menus

PC application menus and menu items:

File

Import File	- Loads memory arrays from hex file
Export File	- Saves memory arrays into hex file
-	
1 C:\TEMP\project.hex	- up to 4 hex file "history" documents
-	
Exit	- Quits Application

Device Family

< Dynamic>	- menu generated when loading the Device File. Will have 1 entry for each device family supported. Clicking a family entry looks for an attached device in that family.

Programmer

Read Device	- Reads memory from attached device.
Write Device	- Performs Chip Erase & writes GUI memory arrays to attached device.
Verify	- Compares device memory with GUI memory arrays.
Erase	- Performs Chip Erase.
Blank Check	- Compares device memory with blank memory arrays.
-	
Verify On Write	- Check/Uncheck. If checked, verifies immediately after write.
Hold Device in Reset	- Check/Uncheck. When checked, sets nMCLR/VPP pin = GND. When unchecked, tri-states the pin.
Write on PICkit Button	- Check/Uncheck. When checked, a Write can be initiated by pressing the PICkti 2 button.
-	
Manual Device Select	- Sets the "Part Detect" property of all families to False
PICkit 2 Programmer-To-Go	- Opens the Programmer-To-Go wizard

Tools

Enable Code Protect	- Enables Code protect bits in configuration.
Enable Data Protect	- Enables Data protect bits in configuration.
OSCCAL	- Only enabled for devices with an OSCCAL instruction at the end of Program Memory.
Set Manually	- Erases device and sets OSCCAL instruction to given value.
Auto Regerate	- Downloads and runs a calibration program on the target PIC
Target VDD Source	- (submenu)
Auto-Detect	- PICkit 2 checks for powered/unpowered target on each operation.
Force PICkit 2	- PICkit 2 always attempts to power the device
Force Target	- PICkit 2 always assumes the target is self-powered.
Calibrate VDD & Set Unit ID…	- Opens a wizard to allow calibration of PICkit 2 voltages, and set a Unit ID (Identifying string for PICkit 2 unit.)
Use VPP First Program Entry	- For applicable families, use an alternate Program Entry script which applies VPP before VDD
Fast Programming	- When unchecked, uses slower programming.
-	
UART Tool	- Switches the application to the UART tool GUI.
Logic Tool	- Switched the application to the Logic Tool GUI
-	
Check Communication	- Looks for PICkit 2 and a target device.
Troubleshoot…	- Opens a wizard to exercise the PICkit 2 pins for operational and connectivity troubleshooting.
-	
Download PICkit 2 Operating System	- Downloads firmware to PICkit 2 using bootloader

View

Single Window	- The default & legacy view format
Multi-Window	- Displays memories in separate windows. Enables following:
-	
Show Program Memory	- Toggles whether Program Memory window is displayed or not
Show EEPROM Data	- Toggles whether EEPROM Data window is displayed or not
-	
Associate / Memory Displays in Front	- When enabled, memory windows move with main window, and minimize, and come to front together. Memory windows will always display in front of Main window (as they are owned by it)

Help

PICkit 2 User's Guide	- Launches PDF file.
44-Pin Demo Board Guide	- Launches PDF file
LPC Demo Board Guide	- Launches PDF file.
PICkit 2 on the web	- Opens www.microchip.com/pickit2 in a browser
ReadMe	- Launches Readme.txt file.
About	- Pops dialog with license agreement and GUI, Device File, & firmware versions.

7.2 Status Window

The GUI Status window lists the following information, some of which may be only appear for relevant parts. Other times it will not be visible.

Label	Example	Notes
Device	PIC16F629	"Not Present" if no device detected.
User ID's	7F 7F 7F 7F	Visible if *UserIDWords* > 0 Displays 7 LSBs of each word as 2 char hex
Checksum	NNNN	Sum of program memory plus masked config words.
Config Words	1234 1234 1234 1234 1234 1234 1234	Displays up to 8 config words, based on *ConfigWords.* Config1 is upper left, Config 2 is directly to the right. Each word displayed as 4 character hex.
OSCCAL	3444	Visible if *OsccalSave* = 'Y' Displays last word of program memory as 4 char hex.
BandGap	3000	Visible if *BandGapMask* > 0000.
< >	<01>	Revision brackets display revision code for active device (excludes baseline). Only visible if REVS: is added to INI file.

7.3 PC Application Startup

On startup, the PC application performs the following tasks:

1. Look for PICkit2. If not found, display an error and disable all functions. If found, check firmware version is minimum required. If not, disable all functions except to download new firmware.
2. Check the device file version for minimum. If not valid or no device file, display error and disable all functions.
3. Load Device File. When loading the Device File, the PC application loads the families in index order. Each becomes a menu option under Menu "Device Family", with the menu text being *FamilyName*. Any duplicate *FamilyIDs* will overwrite earlier entries. The *FamilyName* parameter is also used to title the status window. Ex "Midrange Device Configuration."
4. Search for Devices. The GUI will search for devices in order of the *SearchPriority* family parameter. If no device is found, the application will default to last the family setting from the INI file.
5. Initialize all memory arrays. Program memory blank value set according to device family found.
6. Set form appropriately for family and device found.

7.4 Hex File Import

Menu option

File -> Import File

On selection, this menu item will:

1. Bring up dialog to select file
2. Load all data with addresses from (0) to (*ProgMem* – 1) into the program memory array.
3. Load all data with addresses from (*EEAdr*) to (*EEAdr* + *EEMem*) into the EE data array.
4. Load all data with addresses from (*UserIDAddr*) to (*UserIDAddr* + *UserIDWords*) into the User ID memory array.
5. Load all data with addresses from (*ConfigAddr*) to (*ConfigAddr* + *ConfigWords*) into the configuration memory array.
6. Update "Source" line on GUI.
7. Close file & clean up.

7.5 Hex file Export

Menu option

File -> Export File

On selection, this menu item will:

1. Bring up dialog to select file name & location.
2. Save data from program memory array to addresses (0) to (*ProgMem* – 1).
3. Save data from the EE data array to addresses (*EEAdr*) to (*EEAdr* + *EEMem*.
4. Save data from the User ID memory array to addresses (*UserIDAddr*) to (*UserIDAddr* + *UserIDWords*).
5. Save data from the configuration memory array to addresses (*ConfigAddr*) to (*ConfigAddr* + *ConfigWords*).
6. Close file & clean up.

7.6 Read Device

Menu option

Programmer -> Read Device

"Read" Button

1. Check for a connected device in the currently selected Device Family. If none found, abort and update status.
2. Read program memory into program memory array using *ProgEntryScript*, *ProgMemRdPrepScrpt*, *ProgMemRdScript*, and *ProgExitScript*.
3. If *EEMem* > 0, read EE data in to EE data array using *ProgEntryScript*, *EERdPrepScript*, *EERdScript*, and *ProgExitScript*.
4. If *UserIDWords* > 0, read User IDs using *ProgEntryScript*, *UserIDRdPrepScript*, *UserIDRdScript*, and *ProgExitScript*.
5. If *ConfigWords* > 0, read configuration words using *ProgEntryScript*, *ConfigRdPrepScript*, *ConfigRdScript*, and *ProgExitScript*.
6. Update status & "Source" line on GUI.

7.7 Write Device

Menu option

Programmer -> Write Device

"Write" Button

1. Check for a connected device in the currently selected Device Family. If none found, abort and update status.
2. Check for *VddErase*. If Vdd below, pop warning allowing user to continue or cancel.
3. Check for date-time change on hex file, if loaded. If so, reload file.
4. If *OsscalSave* = 'Y', read OSCCAL from *ProgMem*-1 using *ProgEntryScript*, OSCCALRd*Script*, and *ProgExitScript*.
5. Store OSCCAL into last word of program memory array.
6. If *BandGapMask* > 0, read BandGap config word using *ProgEntryScript*, *ConfigRdPrepScript*, *ConfigRdScript*, and *ProgExitScript*.
7. Store BandGap bits in first word of configuration array.
8. Perform a chip erase using *ProgEntryScript*, *ChipEraseScript*, and *ProgExitScript*
9. Going backwards from *ProgMem* in the program memory array, find the last non-blank memory address.
10. Write program memory array contents up to the last address found in (4) using *ProgEntryScript*, *ProgMemWrPrpScrpt*, *ProgMemWrScript*, and *ProgExitScript.* A 3-byte address is always sent as the first bytes in the download buffer for each execution of the script.
11. If *EEMem* = 0, skip steps 11 & 12.
12. Going backwards from *EEMem* in the EE data array, find the last non-blank EE byte address.
13. Write EE data array contents up the address found in (11) using *ProgEntryScript*, *EEWrPrepScript*, *EEWrScript*, and *ProgExitScript.*
14. If *UserIDWords* = 0, skip step 15.
15. Write User ID words using *ProgEntryScript*, *UserIDWrPrepScript*, *UserIDWrScript*, and *ProgExitScript.*
16. Verify device Code, EE, & UserIDs.
17. If verify fails, display error section & address and abort.
18. If menu Tools->Code Protect Device is selected, apply *CPMask* to *CPConfig* to enable all CP bits
19. If (*ConfigWords* = 0) OR (*ConfigAddr* < *ProgMem*), skip step 20.
20. Write configuration words using *ProgEntryScript*, *ConfigWrPrepScript*, *ConfigWrScript*, and *ProgExitScript.*
21. Display "write successful"

7.8 Verify Device

Menu option

Programmer -> Read Device

"Read" Button

1. Check for a connected device in the currently selected Device Family. If none found, abort and update status.
2. Read part code memory up to *VerifyStop* and compare to program memory array using *ProgEntryScript*, *ProgMemRdPrepScrpt*, *ProgMemRdScript*, and *ProgExitScript.*
3. If a mismatch is found, display error and memory address and abort.
4. If *EEMem* = 0, skip steps 5 & 6.
5. Read part EE data and compare to EE data array using *ProgEntryScript*, *EERdPrepScript*, *EERdScript*, and *ProgExitScript.*
6. If a mismatch is found, display error and EE address and abort.
7. If *UserIDWords* = 0, skip steps 8 & 9
8. Read User IDs using and compare to User ID array using *ProgEntryScript*, *UserIDRdPrepScript*, *UserIDRdScript*, and *ProgExitScript*
9. If a mismatch is found, display error and abort.
10. If (*ConfigWords* = 0) OR (Verify Device is called from Write Device) skip steps 11 & 12.
11. Read configuration words and compare to config array using *ProgEntryScript*, *ConfigRdPrepScript*, *ConfigWrScript*, and *ProgExitScript.*
12. If a mismatch is found, display error and abort.
13. Display "verify successful"

7.9 Erase Device

Menu option

Programmer -> Erase

"Erase" Button

1. Check for a connected device in the currently selected Device Family. If none found, abort and update status.
2. Perform a chip erase using *ProgEntryScript*, *ChipEraseScript*, and *ProgExitScript*
3. Set all memory arrays to erased values (*BlankValue* for program memory array).
4. Set "Source" line to "None"

7.10 Blank Check

Menu option

Programmer -> Blank Check

"Blank Check" Button

1. Check for a connected device in the currently selected Device Family. If none found, abort and update status.
2. Read part code memory up to *VerifyStop* and compare to *BlankValue* using *ProgEntryScript*, *ProgMemRdPrepScrpt*, *ProgMemRdScript*, and *ProgExitScript.*
3. If a mismatch is found, display "Program Memory not blank" and abort.
4. If *EEMem* = 0, skip steps 5 & 6.
5. Read part EE data and compare to 0xFF using *ProgEntryScript*, *EERdPrepScript*, *EERdScript*, and *ProgExitScript*
6. If a mismatch is found, display "Program Memory not blank" and abort.
7. If *UserIDWords* = 0, skip steps 8 & 9
8. Read User IDs using and compare to User ID array using *ProgEntryScript*, *UserIDRdPrepScript*, *UserIDRdScript*, and *ProgExitScript*
9. If a mismatch is found, display "User IDs not blank" and abort.
10. If (*ConfigWords* = 0) skip steps 11 & 12.
11. Read configuration words and compare to *ConfigMasks* using *ProgEntryScript*, *ConfigRdPrepScript*, *ConfigWrScript*, and *ProgExitScript.*
12. If a mismatch is found, display "Configuration not blank" and abort.
13. Display "Device is Blank."

7.11 INI file

The PC Application uses a text .ini file. This file will be read at startup, and saved when the application is closed. If no file exists at startup, then all parameters will default. (The file may simply be deleted when the application is closed to restore all settings to defaults.)
The parameters in the file allow various GUI options to be "remembered" from one session to another.
Comments are indicated with a semicolon. The following comments will be present at the start of the file: Application version, date & time saved.

Parameters:
All parameters are four characters followed by a colon. The colon is followed by a space and the value.

ADET: <Y, N>
- 'Y' (default) to auto-detect parts of applicable families
- 'N' to disable auto-detect, and select all parts manually.

PDET: <Y, N>
- 'Y' (default) to auto-detect parts on starting application
- 'N' to disable auto-detect on startup (If ADET = N, this is set to N)

LFAM: <family string from device file>
- last family used. If no part is found on startup, will default to LFAM

VRFW: <Y, N>
- 'Y' if verify on write menu option is checked.

WRBT: <Y, N>
- 'Y' if write on button menu option is checked.

MCLR: <Y, N>
- 'Y' if Hold Device in Reset menu option is checked.

TVDD: <Auto, PICkit, Target>
- which Target VDD Source option is selected.

FPRG: <Y, N>
- 'Y' if Fast Programming is checked.

PCLK:< 2 - 16>
- SET_ICSP_SPEED argument for "slow" (Fast Programming unchecked)

PASC: <B, Y, N>
- 'Y' if program memory view is hex & word ASCII
- 'B' if program memory view is hex & byte ASCII

EASC: <B, Y, N>
- 'Y' if EEPROM memory view is hex & word ASCII
- 'B' if program memory view is hex & byte ASCII

EDIT: <Y, N>
- 'Y' to allow editing of program and EE memory data.

REVS: <Y>
- When label exists, it enables display of device revision <nn> in Status Window. Delete label to disable.

SETV: <x.y>

- VDD set value; only gets restored on startup where LFAM is the startup family (ie not if a part in a different family is detected). 'x' is volts, 'y' is tenths of volts.

CLBF: <Y, N>

- 'N' if "Programmer -> Clear Memory Buffers on Erase" is unchecked.

HEX1: <path to hex file>
- first hex file history. Blank if none.

HEX2: <path to hex file>
- second hex file history. Blank if none.

HEX3: <path to hex file>
- third hex file history. Blank if none.

HEX4: <path to hex file>

- last hex file history. Blank if none.

TMEN: <Y, N, - >
- When this label exists, it enables Test Memory
- 'Y' open Test Memory window on startup
- 'N' or no argument: do not open Test Memory window on startup.

TMWD: <16 – 1024>
- Only has meaning if TMEN present
- Number of Test Memory words

TMIE: <Y, N>
- Only has meaning if TMEN present
- State of Test Memory Import-Export Checkbox.

SDAT: <Y, N>
- 'Y' on startup, display a device file selection dialog
- 'N' or label doesn't exist: do not open selection dialog, use "PK2DeviceFile.dat"

The following parameters are for the UART Tool:
UABD: <baud rate string>

- Last selected baud rate. Blank if none.

UAHX: <Y, N>

- Y if in Hex mode, N if ASCII mode.

UAS1: <macro string 1>

- Macro/Hex string. Blank if none.

UAS2: <macro string 2>

- Macro/Hex string. Blank if none.

UAS3: <macro string 3>

- Macro/Hex string. Blank if none.

UAS4: <macro string 4>

- Macro/Hex string. Blank if none.

UACL: <Y, N>
- Y if Append CR & LF checked.

UAWR: <Y, N>
- Y if Wrap checked.

UAEC: <Y, N>
- Y if Echo On checked.

These parameters are for window views:
MWEN: <Y, N>
- Y if Multi Window view enabled.

MWLX: <0 – 9999>
- Location of the main window, X co-ord.

MWLY: <0 – 9999>
- Location of the main window, Y co-ord.

MWFR: <Y, N>
- Y to set up multi-window forms so they are associated and memory displays always in front.
- N to set up to main window is always behind.

PMEN: <Y, N>
- Y if program memory window enabled (open).

PMLX: <0 – 9999>
- Location of the program memory window, X co-ord.

PMLY: <0 – 9999>
- Location of the program memory window, Y co-ord.

PMSX: <0 – 9999>
- Size of the program memory window, X co-ord.

PMSY: <0 – 9999>
- Size of the program memory window, Y co-ord.

EEEN: <Y, N>
- Y if eeprom memory window enabled (open).

EELX: <0 – 9999>
- Location of the eeprom memory window, X co-ord.

EELY: <0 – 9999>
- Location of the eeprom memory window, Y co-ord.

EESX: <0 – 9999>
- Size of the eeprom memory window, X co-ord.

EESY: <0 – 9999>
- Size of the eeprom memory window, Y co-ord.

These parameters are for the Logic Tool:

LTAM: <Y, N>
- Y if analyzer is active mode

LTZM: <0 – 3>
- Zoom level

LTT1: <0 – 5>
- Channel 1 trigger setting

LTT2: <0 – 5>
- Channel 2 trigger setting

LTT3: <0 – 5>
- Channel 3 trigger setting

LTTC: <1-256>
- trigger count setting

LTSR: <0 – 7>
- sample rate setting

LTTP: <0 – 5>
- trigger position

LTCE: <Y, N>
- Y if cursors are enabled

LTCX: <0 – 4095>
- X cursor position.

LTCY: <0 – 4095>
- Y cursor position.

The following parameters are for Programmer-To-Go:
PTGM: <0 – 1>
- 0 = 128K PICkit 2 unit
- 1 = 256K upgraded PICkit 2 unit

8.0 Device File

The device file is divided into 4 main sections. The first section gives general information about the device file itself. The second section lists and defines device families, which contain parameters common to all parts in the family. The third section contains parameters for each supported part, including which family a part belongs to. The final section defines scripts that may be referred to in each part's parameters.

NOTE: This section contains a listing of parameters and descriptions of their function. It is not intended to describe the binary file format or order of parameters in such file. See the Source Code for file structure information.

8.1 Device File Parameters

Parameter Name	**Range**	**Examples**	**Description**
VersionMajor	0-99	'1'	Device File major version
VersionMinor	0-99	'0'	Device File minor version.
VersionDot	0-99	'2'	Device File development version.
VersionNotes	String	'Initial Release'	Text notes for version
NumberFamilies	Integer	'5'	The number of Family Parameter sets in the device file.
NumberParts	Integer	'187'	The number of Part Parameter sets in the device file
NumberScripts	Integer	'170'	The number of Scripts in the device file.
Compatibility	0-255	'1'	A number that allows the PC Application to determine if it can use this Device File version.

8.2 Family Parameters

Parameter Name	Range	Examples	Description
FamilyID	0-255	‘1’	Each family has a unique ID that is used to reference it in the Part Parameter section.
FamilyType	0-255	‘3’	This is actually used as the “Device Family” menu display order in the GUI, with ‘0’ being the first displayed family.
SearchPriority	0-255	‘2’	This is the order in which the GUI will search for parts on startup. Lower numbers first.
FamilyName	String	‘PIC18_J_’, ‘Baseline’, ‘EEPROMS/24LCxx’	The text appears both on the GUI and as a menu selection under “Device Family”. “submenus” of families can be created under the “Device Family” menu by using a “/” slash to separate the submenu name from the family name.
Vpp	float	“12.0”	Vpp voltage. If set to zero (0), then Vpp = VDD.
ProgEntryScript	0-65535	‘1’	Number of script to enter programming mode.
ProgEntryVPPScript	0-65535	‘1’	Number of script to enter programming mode VPP first.
ProgExitScript	0-65535	‘2’	Number of script to exit programming mode.
ReadDevIDScript	0-65535	‘5’	Number of script used for reading the device ID.
DeviceIDMask	Up to 8 hex characters	‘3FF0’	Mask of significant bits in Device ID word. Usually masks out the version bits.
BlankValue	Up to 8 hex characters	‘3FFF’	The value of a word in an erased part.
BytesPerLocation	0-255	‘2’	# bytes per memory array location.
AddressIncrement	Byte	‘2’	Address increment per array location.
ProgMemHexBytes	byte	‘4’	# bytes in hex file per array location.
EEMemHexBytes	byte	‘1’	# bytes in hex file per array location.
EEMemBytesPerWord	byte	‘2’	# bytes per EE location for script data handling purposes (ex 16F is 2, 18F is 1)
EEMemAddressInc	byte	‘1’	For display purposes, the address per location (word).
UserIDHexBytes	byte	‘2’	# bytes per UserID location in the hex file.
UserIDBytes	byte	‘2’	# bytes per User ID location in part.
ProgMemShift	byte	‘1’	# bits positions to left shift program memory word before downloading, and right shift after uploading. (for Midrange, which data includes start/stop bits.)
PartDetect	Y/N	Y	Autodetects a part and verifies the ID when an operation is selected if ‘Y’. When ‘N’, detection and ID checking is disabled. Instead a combo box list of available parts is displayed.

TestMemoryStart	Unsigned Integer		*Not presently used*
TestMemoryLength	Unsigned Integer		*Not presently used*

8.3 Part Parameters

There should be one "default" parameter set for each device family. This part will have a PartID of '0000' and a PartName of "Not Present."
Any script number value of '0' will be interpreted as the part does not use that script (no script assigned).

Parameter Name	Range	Examples	Description
PartName	20 chars max	'PIC18F24J10'	The name of the part to appear in the GUI
Family	0-255	'4'	This part uses the common parameters defined in the family section. (points to a Family Parameter set FamilyID)
DeviceID	Up to 8 hex characters	'1D00'	The part's unique device ID. Rev bits are set to zero.
ProgramMem	0 to 262144 Words	'8192'	Size of program memory in words.
EEMem	0 to 4096 Words	'256'	Size of integrated EEPROM in words. (8bit = 1byte/word, 16b = 2bytes/word) '0' indicates no EEPROM.
EEAddr	unsigned int	'2100'	Starting address of EE data in a hex file.
ConfigWords	0-255	'4'	Number of configuration words.
ConfigAddr	unsigned int	'1FFC'	Starting address of configuration words in a hex file. If less than ProgMem, the words will also be placed in the Program Memory array.
UserIDWords	0 to 255	'0'	Number of User ID words. '0' indicates no User ID words.
UserIDAddr	unsigned int	'2000'	Starting address of User ID words in a hex file.
BandGapMask	Up to 8 hex characters	'3000'	A mask for Config bits to be saved across erases. Always applies to first config word. If 0000, then it is ignored.
ConfigMasks	Up to 8 hex character words	'04E1.0FC7.0100.0000'	Mask for configuration bits. The left-most word is the least significant config word. The number of words should match *ConfigWords* **For 10F/12F/16F Devices with "OsccalSave = Y"** Word 8 is used as an OSSCAL mask to verify a valid instruction. Ex "0x0C00" for baseline devices. Word 7 is used as the configuration value during a "regenerate OSCCAL" operations. **For Baseline Devices** Word 6 is used as a configuration word "OR" mask, to set bits on Import (for

			16F5x devices.) **For PIC18F Devices:** The last Config mask (mask8) is used as a “Programmer-To-Go” checksum adjustment for checksum verification to account for masked off but active configs bits in some devices. For **Serial EEPROM Devices**, *ConfigWords* should be set to zero. The words have the following meaning: Word1 : 1 = I2C, 2=SPI, 3= Microwire, 4 = UNIO Word2 : address bytes mask Word3 : number address bits Word4 : I2C number active Ax pins
ConfigBlank	Up to 8 hex character words	‘04E1.0FC7.0100.0000’	Value of the configuration words after a Chip Erase has been performed. The number of words should match *ConfigWords*
CPMask	Up to 8 hex characters	‘0400’	Mask for the code protect bits in the config word given by *CPConfig*.
CPConfig	0-255	‘3’	The number of the configuration word to which *CPMask* applies.
OsccalSave	Y / N	‘N’	If ‘Y’, then the last program memory location is saved across erase cycles.
IgnoreAddress	Unsigned int		Starting Address (in hex file) of section of hex file to be ignored – For sections that aren’t used so they don’t generate an “Hex file too large” warning.
IgnoreBytes	ushort		Starting from IgnoreAddress the number of hex file bytes to ignore.
VddMin	2.5 – 5.0	‘2.5’	Minimum Vdd supported for the part.
VddMax	2.5 – 5.0	‘5.0’	Maximum Vdd supported for the part.
VddErase	2.5 – 5.0	‘4.5’	Minimum Vdd supported for bulk erases.
CalibrationWords	0-255		Number of calibration words following config words in configuration memory.
ChipErasePrepScript	0-65535		Script run before Chip Erase Script if preset (non-zero)
ChipEraseScript	0-65535	‘12’	Number of script used to perform a chip erase.
ProgMemAddrSetScript	0-65535	‘31’	Number of script for setting Program Memory address.
ProgMemAddrBytes	1-256	‘3’	Length of address sent down in bytes for ProgMemAddrSetScript
ProgMemRdScript	0-65535	‘13’	Number of script used to read program memory.
ProgMemRdWords	1-256	‘32’	The number of words read by one execution of the script.
EERdPrepScript	0-65535	‘32’	Number of script intended as one-time setup or preparation for reading EE Data.
EERdScript	0-65535	‘15’	Number of script used to read EEPROM data.

EERdLocations	1-256	'4'	The number of memory locations read by one execution of the script.
UserIDRdPrepScript	0-65535	'33'	Number of script intended as one-time setup or preparation for reading UserIDs.
UserIDRdScript	0-65535	'16'	Number of script used to read the UserID words. Should return *UserIDWords* number of words.
ConfigRdPrepScript	0-65535	'34'	Number of script intended as one-time setup or preparation for reading configuration words.
ConfigRdScript	0-65535	'17'	Number of script used to read Configuration words. Should return *ConfigWords* number of words.
ProgMemWrPrepScript	0-65535	'35'	Number of script intended as one-time setup or preparation for writing program memory.
ProgMemWrScript	0-65535	'20'	Number of script used to write program memory.
ProgMemWrWords	1-256	'32'	The number of words written by one execution of the script.
ProgMemPanelBufs	1-256	'4'	*Not presently used*
ProgMemPanelOffset	unsigned int	'4096'	*Not presently used*
EEWrPrepScript	0-65535	'36'	Number of script intended as one-time setup or preparation for writing EE data.
EEWrScript	0-65535	'21'	Number of script used to write EEPROM data.
EEWrLocations	0-255	'1'	The number of locations written by one execution of the script. Bytes for 8bit, words for 16-bit
UserIDWrPrepScript	0-65535	'37'	Number of script intended as one-time setup or preparation for writing UserIDs.
UserIDWrScript	0-65535	'22'	Number of script used to write the UserID words. Should write *UserIDWords* number of words.
ConfigWrPrepScript	0-65535	'37'	Number of script intended as one-time setup or preparation for writing configuration words.
ConfigWrScript	0-65535	'23'	Number of script used to write Configuration words. Should write *ConfigWords* number of words.
OSCCALRdScript	0-65535	'30'	Number of script used to read OSCCAL byte.
OSCCALWrScript	0-65535	'31'	Number of script used to write OSCCAL byte (after erase).
ProgMemEraseScript	0-65535	'32'	Number of script to erase program memory only
EEMemEraseScript	0-65535	'33'	Number of script to erase EE memory only
ConfigMemEraseScript	0-65535	'34'	Number of script to erase configuration memory only.
DPMask	16-bit		Mask for the EE data protect bits in the config word given by *CPConfig*.
WriteConfigOnErase	Y/N		Some parts (some 18F, dsPIC33)do not erase Config to default values on Bulk

			Erase, so default config values must be written on erase
BlankChkSkipUsrIDs	Y/N		Some parts (dsPIC33) do not erase User IDs to blank on Bulk Erase, so skip them on Blank Check.
TestMemoryRdScript	0-65535		Number of script to read words from Test Memory
TestMemoryRdWords	0-65535		# words read by TestMemoryRdScript
EERowEraseScript	0-65535		Script for row erasing EE Data memory (needed for dsPIC30). Used for low Vdd programming. Note script advances PC by EERowEraseWords on each execution.
EERowEraseWords	0-65535		Number of words (locations) erased by one executiong of EERowEraseScript

ExportToMPLAB	Y/N		Meta-data only used by Device File Editor
DebugHaltScript			*Only used by MPLAB*
DebugRunScript			*Only used by MPLAB*
DebugStatusScript			*Only used by MPLAB*
DebugReadExecVerScript			*Only used by MPLAB*
DebugSingleStepScript			*Only used by MPLAB*
DebugBulkWrDataScript			*Only used by MPLAB*
DebugBulkRdDataScript			*Only used by MPLAB*
DebugWriteVectorScript			*Only used by MPLAB*
DebugReadVectorScript			*Only used by MPLAB*
DebugRowEraseScript	0-65535		Script for row erasing program memory. Used for low Vdd programming. Note script advances PC by DebugRowEraseWords on each execution.
DebugRowEraseWords	0-65535		
DebugReserved5		(Emulation Wr)	*Only used by MPLAB*
DebugReserved6		(Emulation Rd)	*Only used by MPLAB*

8.4 Script Definitions

Parameter Name	Range	Examples	Description
ScriptNumber	1-65536	‘121’	Essentially, its array index number + 1 (0 reserved for no script)
ScriptName	String		Text name of the script
ScriptVersion	0-65535	‘2’	Version of the script itself. Should be incremented on changes.
ScriptLength	0-61	‘37’	Length of script in terms of array elements. Presently must not exceed 61.
Comment	String		Text comments related to the script
Script	ushort Array	Array containing script elements (control bytes and arguments). The first element to execute is in array position [0]. The control bytes and arguments are stored in the low byte of each array element. The upper byte is used for tags: 0xAA = Control Byte (as opposed to argument byte) 0xBB = Display argument byte as hexadecimal 0x00 = Display argument byte as decimal	