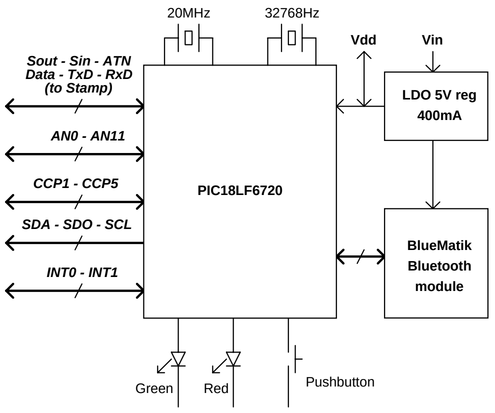
FlexiPanel

ToothPIC Stamp Edition™

Slave microcontroller with Stamp programmer, Bluetooth radio and user interface server

phone and module not to scale



Summary

ToothPIC Stamp Edition is a Stamp-controlled PIC microcontroller with Bluetooth radio, FlexiPanel user interface server and flexible I/O capability. In addition, it can wirelessly in-circuit program the BASIC Stamp from any Bluetooth-equipped PC.

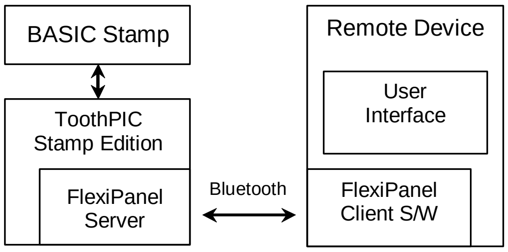
ToothPIC Stamp Edition is ideal for data acquisition and remote control applications where the shallow learning curve of a BASIC Stamp allows fast time-to-market, while the ToothPIC provides flexibility and sophistication needed in electronic products today.

Hardware Features

- *FCC / CE / IC certified Class 1 Bluetooth V1.1 radio, 100m range, integral antenna*
- *128Kbyte Flash, 3.5K RAM, 1K EPROM up to 512Kbyte I2C external memory*
- *12 × 10-bit A to D converter*
- *5 × 10-bit PWM outputs*
- *Serial UART connection to Stamp*
- *20MHz and 32KHz oscillators*
- *Low dropout 400mA power regulator*
- *45 x 22 mm through-hole mount, suitable for breadboards*

Firmware Features

- *Wireless field programming of BASIC Stamps.*
- *FlexiPanel server – creates user interfaces on computers, PDAs, cellphones with no development needed on remote devices.*



- *Stamp commands for:*
  - *System & Bluetooth configuration.*
  - *I/O configuration.*
  - *Setting an output value.*
  - *Reading an input value.*
  - *Managing Bluetooth connections.*
  - *Sending and receiving Bluetooth data.*
  - *FlexiPanel User Interface Server management.*
  - *Sending and receiving user interface control information.*
  - *Reading and writing to memory locations.*
  - *Real time clock control.*

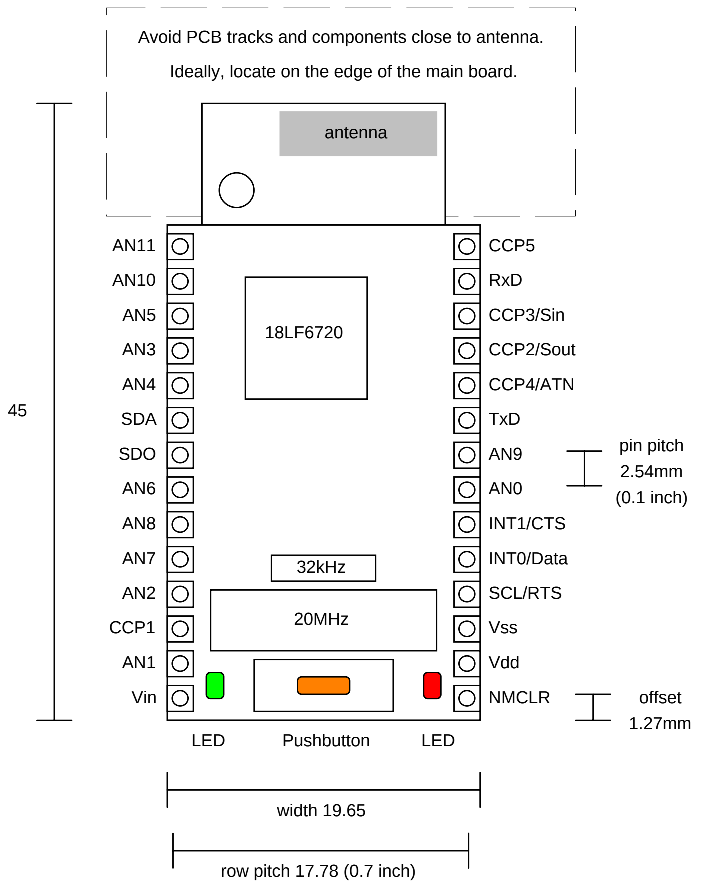
Ordering Information

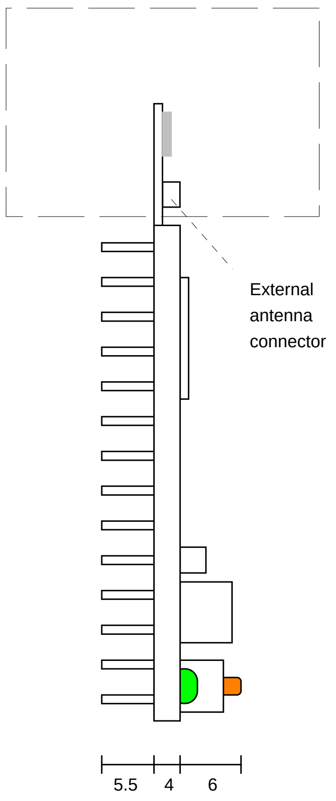
Manufactured to ISO9001:2000

Part No	***Description***
	ToothPIC Stamp Edition 28-pin Dual-in-Line package
	BASIC Stamp Board of Education / BS2p Demo Board Programming Cable

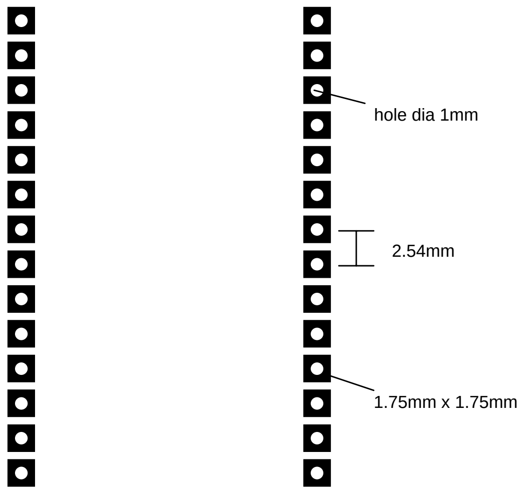
ToothPIC Stamp Edition

Mechanical Data





Side view



Main board PCB pad layout

Dimensions in mm unless otherwise stated

Notes: Ensure the area where the module is mounted has a solid ground plane. To remove the module from an IC socket or breadboard, lever it out using a screwdriver against the pin headers at the sides. Levering from either end may damage components.

Pin Descriptions

Pin Name	***Description***
AN0	Analog input / digital I/O
AN1	Analog input / digital I/O
AN2	Analog input / analog negative voltage reference input / digital I/O
AN3	Analog input / analog positive voltage reference input / digital I/O
AN4	Analog input / digital I/O
AN5	Analog input / digital I/O
AN6	Analog input / digital I/O / Alternate Stamp programmer `ATN` connection
AN7	Analog input / digital I/O / Alternate Stamp programmer `Sin` connection
AN8	Analog input / digital I/O / Alternate Stamp programmer `Sout` connection
AN9	Analog input / digital I/O
AN10	Analog input / digital I/O
AN11	Analog input / digital I/O / Alternate RTS flow control output (note 4)
CCP1	Pulse width modulation output / Alternate RTS flow control output (note 4)
CCP2/Sout	Pulse width modulation output / Stamp programmer `Sout` connection
CCP3/Sin	Pulse width modulation output / Stamp programmer `Sin` connection
CCP4/ATN	Pulse width modulation output / Stamp programmer `ATN` connection
CCP5	Pulse width modulation output
INT0/Data	Data awaiting processing signal / Digital I/O / Alternate RTS flow control output (note 5)
INT1/CTS	Digital I/O / Flow control input from Stamp (note 3)
NMCLR	Reset input (may be left unconnected)
RxD	Serial data input from host controller
SCL	I2C clock / digital I/O / RTS flow control output (note 4)
SDA	I2C data / digital I/O
SDO	Digital I/O / Alternate RTS flow control output (note 4)
TxD	Serial data output to host controller
Vdd	Regulated +5V power input / output (note 1,2)
Vin	Unregulated power input 5 – 10V (note 1,2)
Vss	Power ground reference

1. Either *(i)* regulated power should be provided on Vdd and Vin left unconnected or *(ii)* unregulated power should be provided on Vin and Vdd may be used as a regulated power output. The on-board regulator on the BASIC Stamp is **not** sufficient to power the ToothPIC.

2. If internal regulator is used, total current draw on all outputs (including Vdd if used as a power output) shall not exceed 130mA

3. The Stamp should set this pin to high if it is OK to transmit data to it, low otherwise.

4. The ToothPIC will set this pin to high if it is OK to transmit data to it, low otherwise.

5. The ToothPIC will set this pin to high if there a message waiting to be processed, low otherwise.

Technical Specifications

Physical

Max operating temperature	-20ºC to +75 ºC
Max storage temperature	-30ºC to +85 ºC
Dimensions L × W × H	45mm × 20mm × 10mm excluding pins

Electrical

Supply Voltage (unregulated)	5V to 10V
Supply Voltage (regulated)	4.5V to 5.5V
Peak power requirement excluding draw on I/O pins	270mA
Maximum current on any I/O pin	25mA
Maximum total current on all I/O pins	200mA
Max voltage on I/O pins	-0.5V to +5.5V

Radio

Max RF output power	Class I = 100mW = +20dBm
RF frequency range	2402MHz to 2480MHz
RF channels	79
Frequency hopping	1600 Hz
Range	100m nominal
Maximum data rate	50-90 Kbaud depending on conditions
Pairing method	Unit link key

FCC, CE and IC modular approval

The radio has 'modular approval' for USA, Canada and certain European countries, provided the existing integral antenna is used. The CE mark on the module indicates that it does not require further R&TTE certification. The exterior of the product should be marked as follows:

Contains Transmitter Module FCC ID: CWTUGPZ1
Contains Transmitter Module IC: 1788F-UGPZ1

Quick Start Guide

Introduction

ToothPIC Stamp Edition allows the FlexiPanel Ltd's ToothPIC Bluetooth microcontroller to be managed by a BASIC Stamp via a serial link. While requiring a higher component count than programming the ToothPIC directly, application development is faster, particularly for developers familiar with BASIC Stamp programming. The ToothPIC Stamp Edition is idea for student projects, allowing completion of advanced projects in a short amount of time.

ToothPIC Stamp Edition understands commands for:

- System reset.
- General configuration.
- I/O configuration.
- Setting an output value.
- Reading an input value.
- Managing Bluetooth connections.
- Sending and receiving Bluetooth data.
- FlexiPanel User Interface Server management.
- Sending and receiving user interface control information.
- Reading and writing to memory locations.
- Real time clock control.

Commands can generate responses from ToothPIC Stamp Edition. In addition, unsolicited responses may occur, for example if a user modifies a control on the user interface. Messages are queued up until the Stamp is ready to process them. ToothPIC is intended to respond to binary commands, but it can also accept the same commands if transmitted as ASCII hexadecimal values.

The ToothPIC Stamp Edition can program BASIC Stamps. This is a quite separate mode of operation requiring the use of some of the ToothPIC I/O pins. If the I/O pins are not being used for any other purpose, the programming connections may be left in place during normal operation, as is the case in this quick start guide.

The ToothPIC Stamp Edition consists of the following components:

- 28-pin Dual-In-Line ToothPIC Stamp Edition module.
- Stamp programming cable (4 inch ribbon cable, D9 plug at one end, 3 pin socket at the other).
- ToothPIC Stamp Edition development kit. You need to download this from *www.parallax.com* or *www.flexipanel.com*.

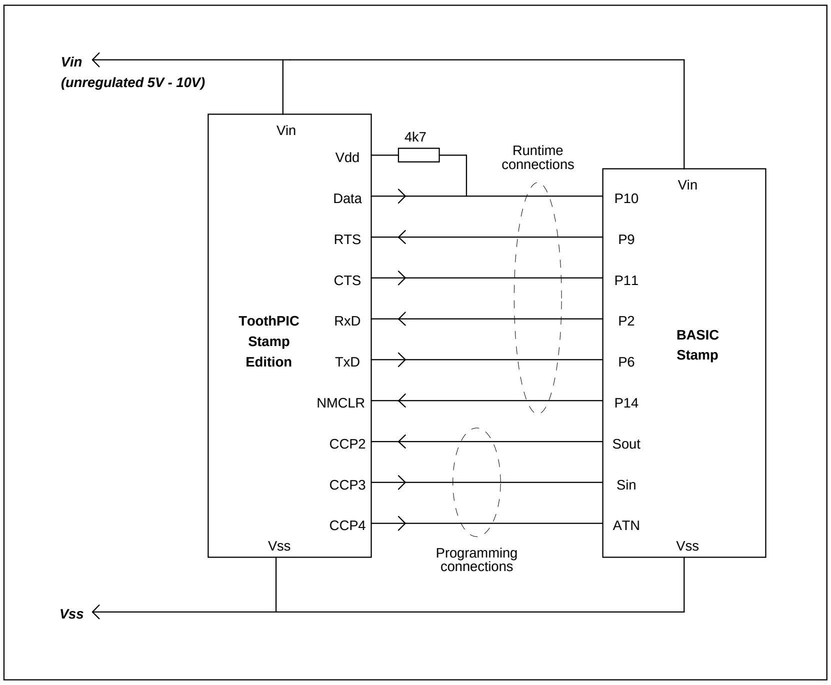
To evaluate the ToothPIC, you will also need:

- A BASIC Stamp. Any Stamp may be used except the BS1 for most parts of the guide. Part *VI* uses multiple slots, so a BS2e, BS2sx, BS2p or BS2pe would be required.
- A prototyping board and a 5-10V power source. A Boards of Education or BASIC Stamp Demo board is ideal.
- A Windows PC with Bluetooth radio. If your PC does not have Bluetooth, low-cost USB adapters are readily available.
- Preferably a 4k7 resistor, a 1k resistor and an LED.

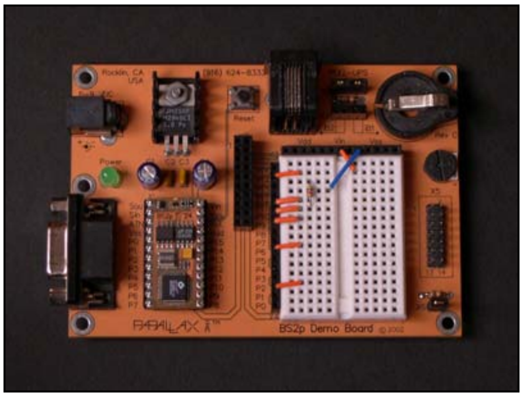
You will also need to know how to use the Bluetooth Manager on your PC to connect to other devices. Since Bluetooth software varies from manufacturer to manufacturer, you will have to refer to the Bluetooth documentation to learn how to do this. In Part *V*, you will be using the HyperTerminal, the terminal emulator bundles with Microsoft Windows, but no prior experience of that program is required.

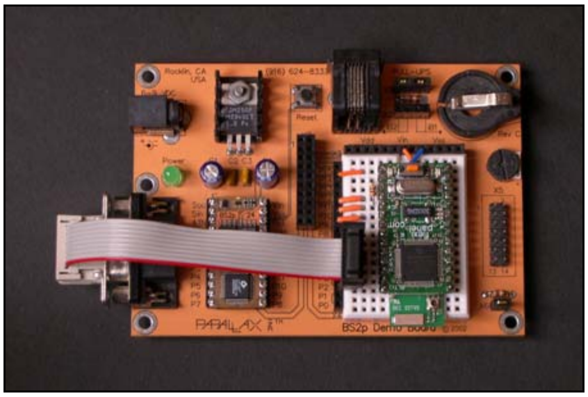
I. Connecting the ToothPIC Stamp Edition to a BASIC Stamp

For the example in the tutorial, connect the ToothPIC Stamp Edition to a BASIC Stamp as shown in the diagram below. If you do not have a 4k7 resistor, that's OK – omit it for the moment remember to take note of the workaround in step 12 later.



The photos below show how you can connect the ToothPIC Stamp Edition on a BS2p Demo Board using the programming cable and a few extra wire links.


Power and runtime connections, 4k7 resistor


ToothPIC and programming cable added

The `Sout` / `Sin` / `ATN` connections are for Stamp programming. The `RxD` / `TxD` / `RTS` / `CTS` / `NMCLR` / `Data` pins are used controlling the ToothPIC at run-time. These are two quite separate sets of connections. If you are only using the ToothPIC for programming, you do not need to wire up the runtime connections. Likewise, if you are not using the ToothPIC for programming, you do not need to wire up the programming connections.

Do not power up the circuit until you are ready to start the tutorials. This is because the existing program on the Stamp may conflict with the way the I/O is connected to the ToothPIC. The ToothPIC may even fail to initialize if it and the Stamp try to output opposing voltages to each other.

While the Stamp is not connected, the red LED may illuminate indicating a serial receive error. The red LED error indicator is self-resetting and will extinguish when the Stamp is connected.

II. Programming the Stamp using ToothPIC Stamp Edition

The first thing to learn is how to program Stamps using the ToothPIC Stamp Edition. Follow these steps:

1. Locate the file `WirelessProg.bsp` in the ToothPIC Stamp Edition Development Kit and open it using the BASIC Stamp Editor. You will see that all the program does is write data to the debug terminal.

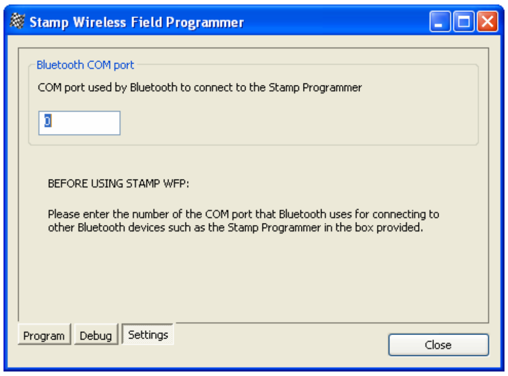
2. If the BASIC Stamp you are using is not a BS2p, change the directive {$STAMP BS2p} so that it refers to the correct type of Stamp, and save it with the correct file extension. You will have to do this for all the examples in this tutorial.

3. Next you will use a program called `Stamp WFP.exe` to program the Stamp. Before you do so, you must check the syntax and save the BASIC file. Type *Ctrl+T* to test check the syntax and then *Ctrl+S* to save the file. Try to get into the habit of typing *Ctrl+T*, *Ctrl+S* before switching to `Stamp WFP.exe`.

4. To prepare the ToothPIC Stamp Edition for wireless field programming, apply power to the ToothPIC with the on-board orange pushbutton held down. Once power has been applied you can release the button. The red and green LEDs will flash simultaneously, indicating that it is waiting for your PC to connect.

   From the Bluetooth Manager application on your PC, search for the device named *ToothPIC SE* and connect to it. If the Bluetooth Manager asks you for a PIN code, you can choose to use no PIN code or use the PIN code '0000'. The Bluetooth Manager will confirm that the connection has been made.

5. Locate and start `Stamp WFP.exe` in the ToothPIC Stamp Edition Development Kit and start it. Note that the file `Tokenizer.dll`, also in the development kit, must be in the same directory. The first time you run it, the screen will look like the screenshot on the right.

   COM port used by Bluetooth to connect to the Stamp Programmer

   0

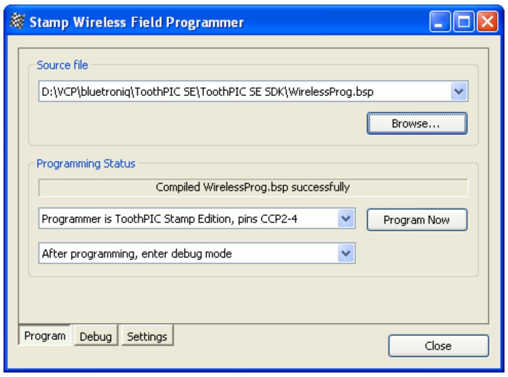
   BEFORE USING STAMP WFP:

   Please enter the number of the COM port that Bluetooth uses for connecting to other Bluetooth devices such as the Stamp Programmer in the box provided.

   Find out which COM port was used by the Bluetooth Manager to connect to the ToothPIC. It probably told you during step 4. Sometimes, this COM port is called the 'Bluetooth outgoing COM port' or the 'Client Applications COM port'. Enter the COM port number in the box provided and then press the *Program* tab.

6. Press the *Browse…* button on the *Program* tab. Locate the `WirelessProg` file you saved in step 3 and select it.

7. Ensure the programming settings are set as shown on the right. You are using a ToothPIC Stamp Edition, and the CCP2-4 pins are going to be used for programming the Stamp.

   D:\VCP\bluetroniq\ToothPIC SE\ToothPIC SE SDK\WirelessProg.bsp

   Compiled WirelessProg.bsp successfully

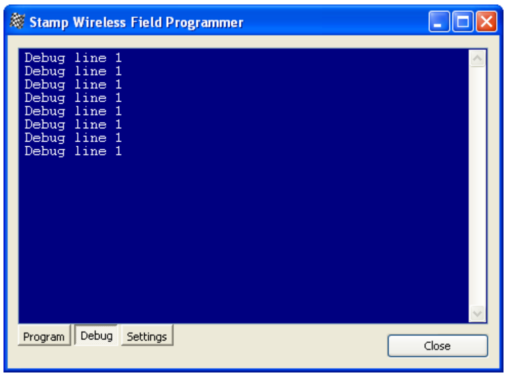
   Programmer is ToothPIC Stamp Edition, pins CCP2-4

   After programming, enter debug mode

   Note the options for what to do after programming. You can enter debug mode immediately or simply wait. In either of these cases, the ToothPIC will stay connected. If you want to reprogram the Stamp, you can re-start from step 8.

   You can also elect that the ToothPIC resets and enters its normal runtime mode. You will likely do this if you are using the Stamp to send commands to the ToothPIC. The Bluetooth connection will be lost, so if you need to reprogram the Stamp you must start again from step 4.

8. Press the *Program Now…* button. When programming is complete, `Stamp WFP` will automatically switch to the Debug tab and debug data will appear.

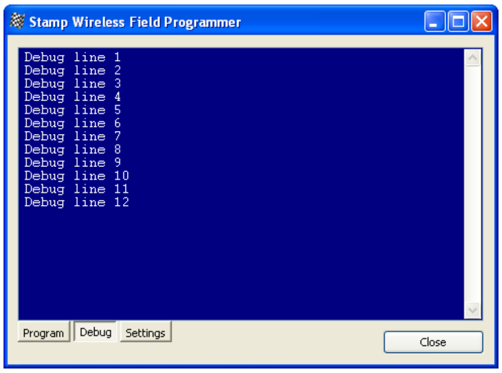
   ```
   Debug line 1
   Debug line 1
   Debug line 1
   Debug line 1
   Debug line 1
   Debug line 1
   Debug line 1
   Debug line 1
   ```

   Note how the debug screen shows 'Debug line 1' being repeated line after line. It was intended to count upwards each line, but the code to increment the counter was omitted. Go back to the Stamp Editor and add the line

   ```
   i = i + 1
   ```

   after the line with the DEBUG command. Press *Ctrl+T*, *Ctrl+S* to verify and save your changes.

9. Back in `Stamp WFP`, switch to the *Program* tab and press the *Program Now…* button again. The Stamp will be reprogrammed and the correct debug messages will be shown as pictured right.

   ```
   Debug line 1
   Debug line 2
   Debug line 3
   Debug line 4
   Debug line 5
   Debug line 6
   Debug line 7
   Debug line 8
   Debug line 9
   Debug line 10
   Debug line 11
   Debug line 12
   ```

   You have now seen how to program BASIC Stamps wirelessly using the ToothPIC Stamp Edition and the `Stamp WFP` application. If that's all you need to do for the moment, then you do not need to follow the rest of the quick start guide. Later, when you want to use the ToothPIC for other purposes such as communicating via Bluetooth, you can start again from the next section.

In this section you have learned how to program Stamps using the wireless field programming feature of the ToothPIC Stamp Edition, how to make changes to your code and re-program the Stamp, and how to use the Field Programmer software to read debug messages.

III. Sending commands to ToothPIC Stamp Edition

In normal use, the Stamp sends commands to the ToothPIC. The remaining parts of this tutorial will demonstrate these commands. A complete explanation of the commands is provided later in the Command / Response Reference section.

The ToothPIC will send responses to all messages sent to it. In addition, it may wish to send unsolicited responses, for example to tell the Stamp that a Bluetooth device has connected to it. Communication is via RxD and TxD serial lines, which the Stamp manages using the `SERIN` and `SEROUT` commands.

The ToothPIC can accept messages at any time, provided the previous message has been processed. When it has been processed, the response is generated. If follow the simple rule that you should wait for the response to the last message before sending another, ToothPIC will always be ready to accept messages.

The BASIC Stamp can only accept responses when the `SERIN` command is being processed. The RTS connection is used to tell the ToothPIC that it has reached a `SERIN` command. The RTS line is usually *high* but switches to *low* when the Stamp is ready to process a response. Unsolicited responses may occur at any time, so the Data connection is used to tell the Stamp when a response has been received. The Data line is usually *low* but switches to *high* when the Stamp is ready to process a response. In addition, it cycles *high* then *low* to indicate that initialization is complete.

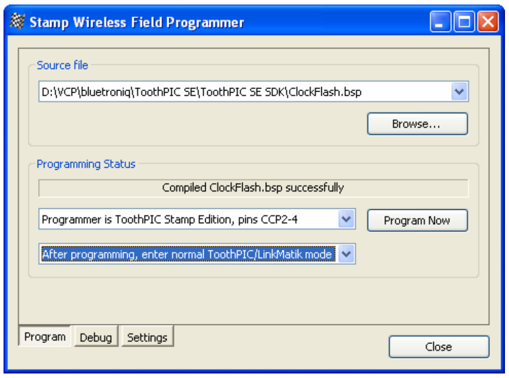
In this section you will learn how to send commands to the ToothPIC and receive responses to those commands. First, the application will be programmed, then the BASIC code will be analyzed. Follow these steps:

10. Locate the file `ClockFlash.bsp` in the ToothPIC Stamp Edition Development Kit and open it using the BASIC Stamp Editor so you can view the code. The application sets the time of the real time clock and then repeatedly asks the time. Each time a second elapses, the green LED is toggled. Each time five seconds elapse, the red LED is toggled. If necessary, change the Stamp directive and re-save the file.

11. In `Stamp WFP`, Press the *Browse…* button on the *Program* tab. Locate the `ClockFlash` file from step 10 and select it.

    Change the After programming option so that the ToothPIC enters its normal mode after programming. If you are not connected to the ToothPIC, connect to it now by applying power with the orange pushbutton pressed, and then connect to it using the Bluetooth Manager.

    Press the Program Now button. After programming, the ToothPIC will disconnect and reset. After re-initializing, it will be under the control of the Stamp. The green LED will toggle every second and the red LED will toggle every five seconds.


Stamp Wireless Field Programmer

Source file: D:\VCP\bluetroniq\ToothPIC SE\ToothPIC SE SDK\ClockFlash.bsp

Programming Status: Compiled ClockFlash.bsp successfully

Programmer is ToothPIC Stamp Edition, pins CCP2-4

After programming, enter normal ToothPIC/LinkMatik mode

12. Examine the BASIC Code in the Stamp Editor. After initializing the I/O pins, the Stamp resets the ToothPIC and waits for initialization to be complete. Alternate commented-our code is shown in case you did not have a 4k7 resistor available. If you were very short of I/O on the Stamp, you could also omit the reset line, but you would have to make sure power was applied to both the Stamp and the ToothPIC to be sure they were in sync.

13. The BASIC Stamp then sets the clock time and date using the following commands:

```
SEROUT TxPinNo, Baud, [ $0B, $01, $42, 00, 30, 22, 20, 05, 05, $D5, $07 ]
SERIN  RxPinNo\RTSPinNo, Baud, [ ch, ch2 ]
```

    The `SEROUT` command sets the time. The format of this and other commands is detailed in the Command / Response Reference section. The `SERIN` command waits for the response from ToothPIC. The response will only ever be the two bytes `$02 $01`, which acknowledge that the command was carried out successfully. The purpose of waiting for the response is to ensure than no further messages are sent until the ToothPIC is ready.

    Note that in all messages and responses, the first byte is always equal to the number of bytes in the message. In this way, both the Stamp and the ToothPIC know how many bytes to receive before processing the information.

14. The main program loop then begins with the lines:

```
SEROUT TxPinNo, Baud, [ $03, $01, $43 ]
SERIN  RxPinNo\RTSPinNo, Baud, [ ch, ch2, sec, mn, hour, date, weekday, month,
                                  year.LOWBYTE, year.HIGHBYTE ]
```

    The `SEROUT` command requests the time. The `SERIN` command waits for the response from ToothPIC. The response will be 10 bytes: the first (`ch`) is the number of bytes in the response, the second is `$43`, indicating that the response is the time and date. The remaining bytes contain the date and time as indicated by the variable names.

15. The remaining lines control the LEDs. Note , for example, the line:

```
SEROUT TxPinNo, Baud, [ $04, $03, $30, $FF ]
```

    which lights the green LED, and:

```
        SERIN  RxPinNo\RTSPinNo, Baud, [ ch, ch2 ]
```

which waits for acknowledgement before sending the next command.

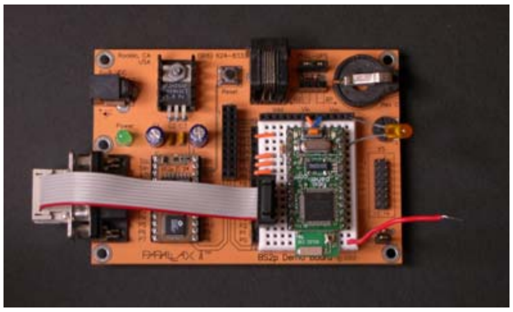
In this section you have learned how to send commands to the ToothPIC. There are commands for ToothPIC configuration, I/O, making Bluetooth connections, running the user interface server and reading and writing to ToothPIC and external memory.

IV. Controlling ToothPIC I/O

The ToothPIC Stamp Edition has a variety of I/O available. In this section you will learn how to configure, read and write I/O. Additionally, the responses from ToothPIC will be examined more closely to detect errors, since, during development, it is easy to send a command which doesn't make sense. Follow these steps:

16. Locate the file `IOComms.bsp` in the ToothPIC Stamp Edition Development Kit and open it using the BASIC Stamp Editor. You will see that the program sets up AN11 as an analog input and CCP1 as a PWM output. In the main loop, the duty cycle of the PWM output is made to vary according to the voltage on the AN11 input. If necessary, change the Stamp directive and re-save the file.

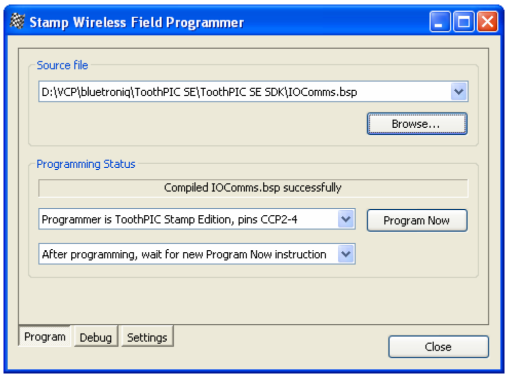
17. To set the voltage input on AN11, you can connect a potentiometer to it to vary the voltage smoothly. (If you have the BS2p demo board, there is a preset pot, as shown in the photo directly under the LED, connected to connector X5 pin 3.) Alternatively a flying lead can be connected to it and you can connect it to Vss, Vdd or even see how the voltage varies as you leave it floating. To monitor the duty cycle on the PWM output, connect an LED to it so that it lights when the voltage is 5V. As the duty cycle increases, the LED will grow brighter. In the photo opposite, a flying lead has been connected to AN11 and an LED has been connected between CCP1 and Vss. The LED has an integral current limiting resistor. To use an LED without an integral resistor, connect a 1K resistor in series.



18. In `Stamp WFP`, Press the *Browse…* button on the *Program* tab. Locate the `IOComms` file from step 16 and select it. Power-up the ToothPIC with the orange pushbutton pressed so that the LEDs flash, and then connect to it using the Bluetooth Manager.

    Press the Program Now button. After programming, Stamp will reset and start executing the code. During initialization, the Stamp resets the ToothPIC.

    Once it has reset, the ToothPIC will repeatedly flash the red LED twice followed by the green LED twice. This is because there was an error in the BASIC code which needs to be corrected.



Stamp Wireless Field Programmer

Source file

D:\VCP\bluetroniq\ToothPIC SE\ToothPIC SE SDK\IOComms.bsp

Programming Status

Compiled IOComms.bsp successfully

Programmer is ToothPIC Stamp Edition, pins CCP2-4

After programming, wait for new Program Now instruction

19. Examine the BASIC Code in the Stamp Editor. After each `SEROUT` command, there are the lines:

```
        DebugVal = xx
        GOSUB GetBytesCheckError
```

    where *xx* is a different value each time. The subroutine `GetBytesCheckError` reads the first two bytes of the ToothPIC response. If the second byte has the value `$02`, the response is an error and

`GetBytesCheckError` will tell ToothPIC to flash the `DebugVal` value (red flashes for tens, green flashes for units.)

Since the flashes indicate the number 22, the error must have occurred processing the line:

```
        SEROUT TxPinNo, Baud, [ $02, $04, $1B ]
```

In this case, the error is that the first byte transmitted, indicating the total number of bytes in the message, is $02 when it should be `$03`. This method of programming error detection is very useful in addition to the Stamp's usual debug facilities, since it is always available, whether or not a debug terminal is connected. It will not, however, detect all programming errors. In particular, if the first byte of a transmit message specifies more bytes than are actually transmitted, ToothPIC will 'hang' waiting for the extra bytes.

20. In the BASIC Stamp Editor, change the `$02` to `$03`, save the file and reprogram the Stamp by repeating step 18. This time, the green light should flash regularly, indicating that the program is functioning correctly.

21. The program reads the analog voltage on the AN11 pin and outputs a PWM duty cycle in proportion to the voltage. If you have connected an LED to the AN11 pin, it will glow in brightness in proportion to the AN11 voltage. Since the analog inputs are very high impedance, the voltage will probably 'float' considerably if unconnected and even touching the input pin will have a noticeable effect on the LED brightness.

In this section you have learned how to read and write ToothPIC I/O. I/O can be configured for binary digital I/O, parallel digital I/O, analog inputs and PWM outputs. In addition, you can control the LEDs and examine the state of the pushbutton.

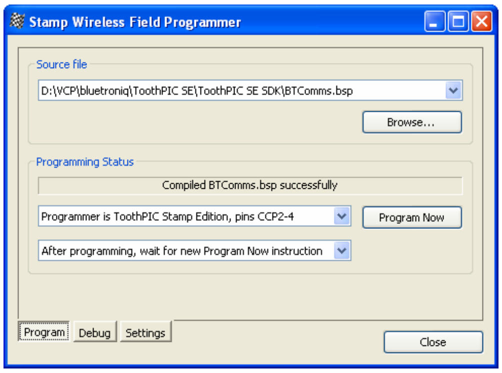
V. Managing Bluetooth communications

The ToothPIC's on-board Bluetooth radio can accept and initiate communications links with other Bluetooth devices. In this section you will learn how to place the radio in *slave mode*, where other devices can connect to it. You will then be able to send it instructions over the Bluetooth link from HyperTerminal, the Windows terminal emulator application. (Slave mode should not be confused with ToothPIC Stamp Edition, which is firmware allowing the ToothPIC to be controlled by microcontrollers other than BASIC Stamps.) Follow these steps:

22. Locate the file `BTComms.bsp` in the ToothPIC Stamp Edition Development Kit and open it using the BASIC Stamp Editor. You will see that the program sets up AN11 as an analog input. In the main loop, the Bluetooth radio is placed in slave mode and it waits for instructions to be sent to it. If it received the ASCII character 'R', it toggles the red LED. If it received the ASCII character 'G', it toggles the green LED. If it received the ASCII character 'V', it measures the voltage on AN11 and reports the value. If necessary, change the Stamp directive and re-save the file.

23. In `Stamp WFP`, Press the *Browse…* button on the *Program* tab. Locate the `BTComms` file from step 22 and select it. Power-up the ToothPIC with the orange pushbutton pressed so that the LEDs flash, and then connect to it using the Bluetooth Manager.

    Press the Program Now button. After programming, Stamp will reset and start executing the code. During initialization, the Stamp resets the ToothPIC. Once it has reset, the ToothPIC will flash the red LED once followed by the green LED five times, indicating error number '15'. An error in the BASIC code needs to be corrected.



Source file: D:\VCP\bluetroniq\ToothPIC SE\ToothPIC SE SDK\BTComms.bsp

Programming Status: Compiled BTComms.bsp successfully

Programmer is ToothPIC Stamp Edition, pins CCP2-4

After programming, wait for new Program Now instruction

24. Examine the BASIC Code in the Stamp Editor. None of the `DebugVal` error values equals 15. This is an internal error generated by the ToothPIC. In this case, it is because ToothPIC is placed in slave mode each pass through the main loop. It should only be placed in slave mode if it is not currently in slave mode. If it is already in slave mode and you instruct it to enter slave mode, error code number 15 is generated. All possible error codes are listed in the Error Codes section.

    In the BASIC Stamp Editor, insert the line

    ```
    IsSlave = 1
    ```

    after the line

    ```
    IF IsSlave=0 THEN
    ```

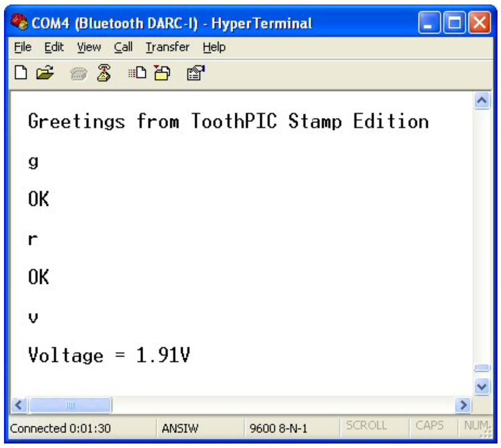
    This ensures that the slave mode command is only sent when the ToothPIC is not in slave mode. Save the file and reprogram the Stamp by repeating step 23. This time, the LEDs should flash alternately and then in unison during initialization, indicating that the program is functioning correctly.

25. The Stamp has instructed the ToothPIC to enter slave mode so that another Bluetooth device can connect to it. We will connect to it using Windows HyperTerminal. When you have completed part *V*, remember to close HyperTerminal. If you fail to do so, it will 'hog' the Bluetooth COM port when you next try to program the Stamp using the Wireless Field Programmer.

    Start the HyperTerminal application on your Windows PC. It's under *Accessories > Communications* in the Windows Start menu. In the initial dialog boxes, give the connection a name and then for *Connect using*, specify the COM port used by Bluetooth to make outgoing connections. You can leave the Port Settings at their default values because the Bluetooth COM port will ignore them.

    Now you need to reconnect to the ToothPIC again using the Bluetooth Manager. Some device Bluetooth device drivers may not notice that the ToothPIC has reset itself and is no longer connected, so the Bluetooth Manager will think it is still connected. However, it isn't and you must reconnect.

26. Once the connection is made, the line *Greetings from ToothPIC Stamp Edition* appears in HyperTerminal. Type an 'R' to toggle the red LED, 'G' to toggle the green LED and 'V' to get a readout of the voltage on the AN11 pin.


COM4 (Bluetooth DARC-I) - HyperTerminal
File Edit View Call Transfer Help
Connected 0:01:30
ANSIW
9600 8-N-1
SCROLL
CAPS
NUM

```
Greetings from ToothPIC Stamp Edition
g
OK
r
OK
v
Voltage = 1.91V
```

    In the BASIC code, you will notice that each of the commands issued by the Stamp after a key press is followed by a call to GetBytesCheckError without checking the actual response received. If you type very quickly, the first response from the Stamp may be more typed data rather than the acknowledgement that the command is complete. So a limitation of the way this example is coded is that you should wait for a response from the Stamp each time before typing your next command. Part VI shows a more sophisticated way of dealing with responses from ToothPIC.

In this section you have learned how to send commands to the ToothPIC to manage the Bluetooth radio, and receive responses when events happen. In addition to allowing devices to connect to it, commands may be issued to scan for other devices, connect to other devices, enter power saving modes and measure the signal strength.

Do remember to close HyperTerminal before moving on to Part *VI*.

VI. Creating user interfaces on remote devices

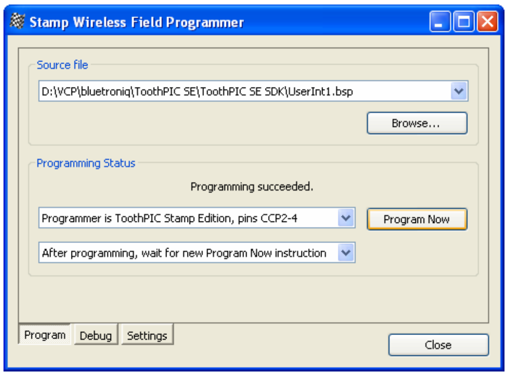
The ToothPIC's FlexiPanel user interface server allows you to create user interfaces on remote devices such as Windows PCs, Pocket PCs and some high-end cellphones. In this section you will learn how to work with the user interface server and how to design and create user interfaces and program them into ToothPIC Stamp Edition.

27. Locate the file `UserInt1.bsp` in the ToothPIC Stamp Edition Development Kit and open it using the BASIC Stamp Editor. You will see that the program does very little other than initialize and then instruct the ToothPIC to start the user interface server. If necessary, change the Stamp directive and re-save the file.

28. In `Stamp WFP`, Press the *Browse…* button on the *Program* tab. Locate the `UserInt1` file from step 22 and select it. Power-up the ToothPIC with the orange pushbutton pressed so that the LEDs flash, and then connect to it using the Bluetooth Manager.

    Press the Program Now button. After programming, Stamp will reset and start executing the code. During initialization, the Stamp resets the ToothPIC.

    Once it has reset, the ToothPIC will flash the red LED then the green LED, then both together.



D:\VCP\bluetroniq\ToothPIC SE\ToothPIC SE SDK\UserInt1.bsp

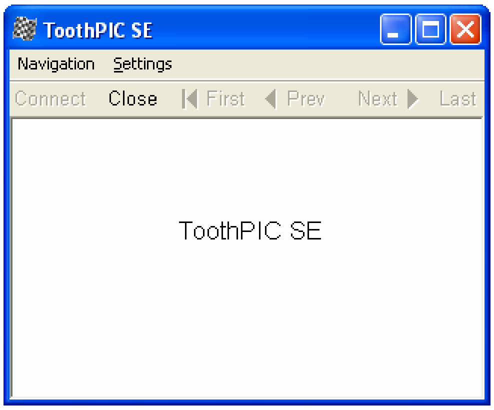
Programming succeeded.

Programmer is ToothPIC Stamp Edition, pins CCP2-4

After programming, wait for new Program Now instruction

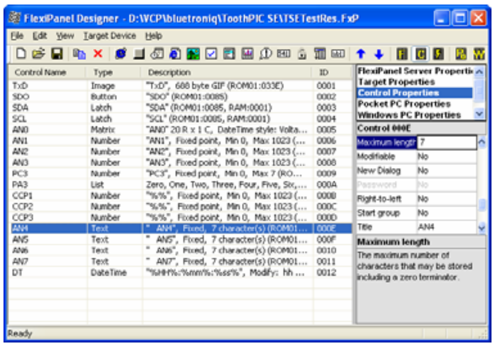
29. To view the user interface on a remote device, you must have FlexiPanel Client software running on that device. Software is available for Windows, Pocket PCs and some high-end cellphones. Download the FlexiPanel Client application of your choice from *www.FlexiPanel.com* and install it.

    Use the Client software to connect to the ToothPIC Stamp edition and default user interface will be displayed. Its appearance will vary according to the client device, but on a Windows PC it looks like the screenshot opposite. The default user interface is a static text control with then words ToothPIC SE in it.



30. The default user interface is a bit boring, so we will program it with a more interesting one. Download the user interface design tool, `Designer.exe`, from *www.flexipanel.com*, and also its documentation `Designer.pdf`. Look through the Quick Visual Tour section so you get a basic idea of how the Designer application works. Run the application and locate and open the file `TSETestRes.FxP`.

    Note that the user interface consists of Image, Button, Latch, Matrix, Number, List, Text and DateTime controls. In this example, they are all in the same 'dialog'. It is possible to have multiple 'dialogs', each with different controls on them. In addition to the controls demonstrated in this example, you can create Blob, Files, Message, Password and Section controls.


FlexiPanel Designer - D:\VCP\bluetroniq\ToothPIC SE\TSETestRes.FxP
File
Edit
View
Target Device
Help

Control Name	Type	Description	ID
TxD	Image	"TxD", 688 byte GIF (ROM01:033E)	0001
SDO	Button	"SDO" (ROM01:0085)	0002
SDA	Latch	"SDA" (ROM01:0085, RAM:0001)	0003
SCL	Latch	"SCL" (ROM01:0085, RAM:0001)	0004
AN0	Matrix	"AN0" 20 R x 1 C, DateTime style: Volta…	0005
AN1	Number	"AN1", Fixed point, Min 0, Max 1023 (…	0006
AN2	Number	"AN2", Fixed point, Min 0, Max 1023 (…	0007
AN3	Number	"AN3", Fixed point, Min 0, Max 1023 (…	0008
PC3	Number	"PC3", Fixed point, Min 0, Max 7 (RO…	0009
PA3	List	Zero, One, Two, Three, Four, Five, Six,…	000A
CCP1	Number	"%%", Fixed point, Min 0, Max 1023 (…	000B
CCP2	Number	"%%", Fixed point, Min 0, Max 1023 (…	000C
CCP3	Number	"%%", Fixed point, Min 0, Max 1023 (…	000D
AN4	Text	" AN4", Fixed, 7 character(s) (ROM01…	000E
AN5	Text	" AN5", Fixed, 7 character(s) (ROM01…	000F
AN6	Text	" AN6", Fixed, 7 character(s) (ROM01…	0010
AN7	Text	" AN7", Fixed, 7 character(s) (ROM01…	0011
DT	DateTime	"%HH%:%mm%:%ss%", Modify: hh …	0012

Control 000E

Maximum length	7
Modifiable	No
New Dialog	No
Password	No
Right-to-left	No
Start group	No
Title	AN4

Maximum length

The maximum number of characters that may be stored including a zero terminator.

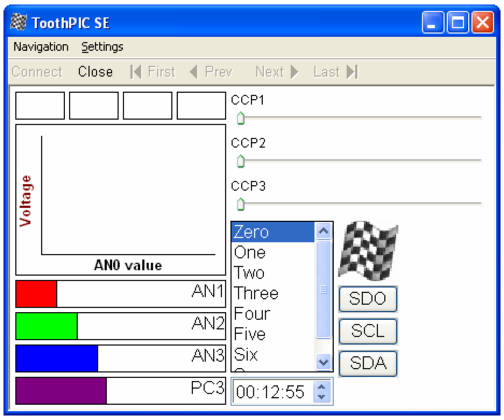
Ready

31. To program the user interface into the ToothPIC, power-up the ToothPIC with the orange pushbutton pressed so that the LEDs flash, and then connect to it using the Bluetooth Manager, just as you did to program the Stamp.

    Ensure *ToothPIC Stamp Edition* is selected as the target device in the Target Device menu. (Do not confuse this with *BASIC Stamp Edition* which is an earlier product.) Select *View > ToothPIC Stamp Edition Properties*, and set the Programming COM port item in the properties list to the Bluetooth COM port used to connect to other devices. Then select *Program ToothPIC Stamp Edition…* from the menu. You will first be asked to save the file `TSETestRes.bsp`. This file contains useful BASIC code for interacting with the controls and you will use it in Part VII. Then it will tell you it is waiting for you to connect to the ToothPIC. When you are connected, press the Program… button. The LEDs should flash as the new user interface is programmed into the Stamp.

32. When programming is complete, reset the Stamp so that it re-starts the ToothPIC and turns on the user interface server. The new user interface will appear. (If you are using the Windows or Pocket PC client, you may need to select *Load Recommended Layout* from the Settings menu because the client is not expecting the user interface to have changed from the last time it connected to the ToothPIC.)

    The appearance on the Windows client software is shown opposite. The Stamp isn't sending any instructions to the ToothPIC yet, so the user interface doesn't do anything just yet except look nice.



In this section you have learned how to send update the user interface that is displayed on the ToothPIC. You have only programmed one dialog. (A dialog is one set of controls that are displayed together.) It is possible to program up to 255 dialogs and select which is displayed at any time. In the next section you will learn how to write commands so that the Stamp can read and write to the controls which are displayed.

VII. Interacting with the user interface

In part VI, you created a new user interface on the ToothPIC. Now you will learn how to interact with it. Interaction will take the form of messages passed to the ToothPIC from the Stamp, and responses from the Stamp. All the possible messages you would like to send, and the expected responses, are documented in the computer generated file `TSETestRes.bsp` which was created by FlexiPanel Designer in step 31. In most cases, you can simply cut and paste from this file to your BASIC program in order to interact with the user interface.

Responses may be received at any time from the ToothPIC, particularly when a user connects, disconnects, or modifies one of the controls. This kind of behavior is requires *event-driven* programming. To achieve this with the Stamp, we will use the POLLRUN command to switch to a slot whenever there in a response to process. The POLLRUN command requires multiple program slots, so you will not be able to use a BS2 for Part VII.

We advise you use a regular RS232 Stamp programming cable (or a LinkMatik SE wireless programmer) for this tutorial. This is because it is not possible to use ToothPIC's wireless debug terminal at the same time as the user interface server.

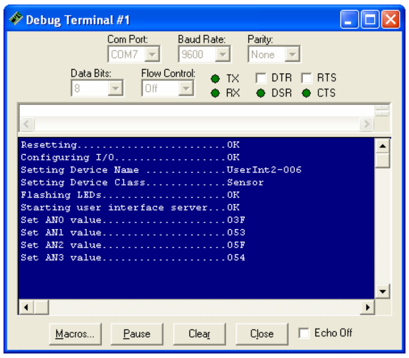
In this section you will learn is how to set up the ToothPIC I/O and then tie those inputs and outputs to controls on the user interface. Roughly once per second, four inputs pins are inspected and their values send to various controls. In addition, then the user modifies some of the controls in the user interface, other output pins are modified. Make sure you have completed part VI correctly, and the follow these steps:

33. Locate the file `UserInt2.bsp` in the ToothPIC Stamp Edition Development Kit and open it using the BASIC Stamp Editor. You will see that four other program slots, `UI2_MainLoop`,

`UI2_ProcResponse`, `UI2_ProcIOVal`, `UI2_ProcCtlVal` are opened, too. This example uses 5 program slots for clarity. In practice, without the extensive debug messages, it could take up fewer slots. The main functions of each slot are as follows:

`UserInt2`	Program initialization
`UI2_MainLoop`	Main program loop for monitoring input values
`UI2_ProcResponse`	'Interrupt' routine to process general ToothPIC messages
`UI2_ProcIOVal`	'Interrupt' routine to process ToothPIC I/O value messages
`UI2_ProcCtlVal`	'Interrupt' routine to process ToothPIC user interface updates

34. If necessary, change the type of BASIC Stamp to suit the Stamp you are using. Then program the Stamp directly from the BASIC Stamp Editor. (You can use the ToothPIC's wireless field programming facility, but then you will not be able to see the debug terminal parts of this tutorial.)

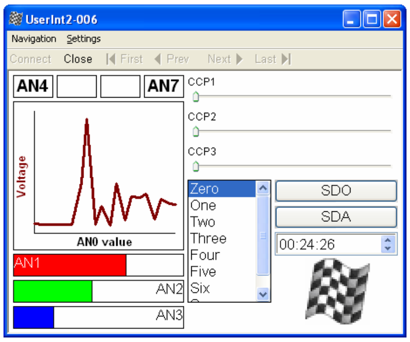


```
Resetting.......................OK
Configuring I/O.................OK
Setting Device Name ............UserInt2-006
Setting Device Class............Sensor
Flashing LEDs...................OK
Starting user interface server...OK
Set AN0 value...................03F
Set AN1 value...................053
Set AN2 value...................05F
Set AN3 value...................054
```

35. Observe in the debug terminal window that the Stamp resets and initializes the ToothPIC, and then starts taking analog to digital readings. It is sending these values to the user interface server for display.

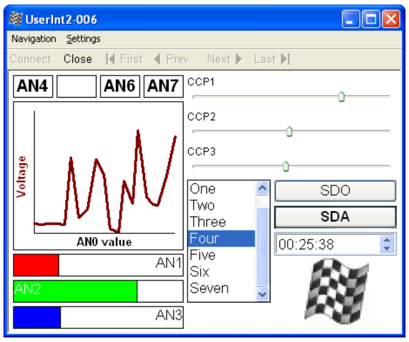
36. Use the Client software to connect to the ToothPIC Stamp edition and default user interface will be displayed. Its appearance will vary according to the client device, but on a Windows PC it looks like the screenshot opposite.

    Note how the chart, progress bar and AN4 to AN7 text controls change over time. Try touching these I/O pins on the ToothPIC to induce stray voltages on them, and observe the changes in the user interface.



37. The list, button, latch and CCP1 to CCP3 controls may be modified. As you change the CCP slider controls, the PWM duty cycle on the respective CCP pins varies. You can see this by connecting LEDs (with current limiting resistors) to the CCP pins. The brightness of the LEDs will reflect the slider positions. Similarly, pressing the SDO button pulses the SDO pin, the SDA latch sets the state of the SDA pin and the list box sets the value of parallel outputs AN9 to AN11.

    The checkered flag image control is an image pushbutton. If you press it, you will be asked if you wish to reset the Stamp.



In this section you have had an overview of how to use the ToothPIC user interface server to interact with the Stamp. The example is quite extensive – it is difficult to provide a 'cut-down' version of an event-driven program for the Stamp. However, do not let that put you off. The example serves as a boilerplate for all event-driven programs you might like to write for the Stamp and ToothPIC SE combination. The next section is explains how the BASIC code works in depth.

Guide to using the User Interface Server

The `UserInt2.bsp` example in part VII of the tutorial is typical of the structure of an event-driven application for user interface processing. This section analyzes the BASIC code from that example to provide an in-depth view of how to write code for user interface processing. There are three main concepts to grasp: *Interrupt and Resume*, *Computer-Generated Code and Constants*, and *Sharing Variables Across Slots*.

Interrupt and Resume

You main program code gets interrupted to process messages as they arrive. Then it has to resume what it was doing. You will require a minimum of two program slots:

Slot A: Performs initialization and then loops through the main program loop.

Slot B: Interrupts Slot A and processes ToothPIC messages as required.

Since ToothPIC may wish to send messages at any time, the easiest way to respond to them is using the `POLLRUN` command to trigger Slot B as soon as a message arrives. When the message has been processed, Slot 1 will start again. Since message arrival will be unpredictable, Slot A needs to keep track of what it was doing so that when it starts again, it can resume from where it was interrupted. You will see in the example code how the variable `NextUpdate` is used to keep track of the next task to perform, and in Slot A there is a `BRANCH` command that uses the `NextUpdate` variable to decide what to do next.

This example actually uses five slots, mostly because of the space required by the `DEBUG` commands:

Slot 0: Performs initialization and then branches to Slot 1.

Slot 1: Main program loop.

Slot 2: Interrupts Slot 1 and processes ToothPIC messages as required.

Slot 3: Called by Slot 2 if the message is an I/O value message.

Slot 4: Called by Slot 2 if the message is a control value or notification message.

Computer-Generated Code and Constants

The messages sent to ToothPIC are binary sequences of varying length. It would be quite easy to get them wrong. For this reason, Designer.exe automatically generates sample code for you. In this example, the sample code file that is generated is `TSETestRes.bsp`. The sample code will not compile itself (it is too big), but you can copy and paste from it as required. The comments in the code will indicate what you need to do.

Each control is referred to using a two-byte Control ID number. Similarly, if you use multiple dialogs, these will be referred to by a single-byte number. These numbers will change as you change your user interface, so you need to update the values in your BASIC code each time you change your user interface. To make this easy, constants are declared in a section between two sets of ========= comment lines. You should cut and paste this entire section when you change your user interface, and in the remainder of your code, you should only refer to the controls by the constants, rather than the numerical values.

Preserving Variables Across Slots

If you wish the value of a variable to be preserved as you switch between slots, it must be declared in every slot and stored in exactly the same place. To do this, you must should declare all such variables in exactly the same order at the beginning of each slot. You must also share any variables which require larger storage space than the variables you wish to share.

In this example, `NextUpdate`, `NumBytes` and `ReplyVal` are byte variables which must be preserved between slots. The word variable `CtlID` also is has to be declared at the beginning of each slot because, although it not shared, is requires more storage than the byte variable being preserved.

Program Initialization

UserInt2.bsp, which is placed in slot 0, initializes the application. As with all other slots, it begins with preserved variable definitions, followed by computer generated constants as discussed earlier. Then, after setting the I/O pin directions, it resets the ToothPIC and waits for the Data pin to go low, indicating that the ToothPIC has initialized.

The first SEROUT command sets up 4 A/D inputs, AN0 to AN3, and is typical of messages sent to ToothPIC Stamp Edition:

```
    SEROUT TxPinNo\CTSPinNo, Baud, [ $04, $02, $01, $04 ]
    GOSUB GetBytesCheckError
```

The first byte of the message, $04, indicates the number of bytes in the message. This is true for all messages and responses. The next byte, $02, is the command to set up I/O pins. The third byte, $01, is the sub-command for setting up A to D inputs. The final byte, $04, indicates that four A/D inputs, AN0 to AN4 should be configured.

Sending this message will generate a response which will either be $02 $01, indicating that the command was executed OK, or $03 $02 $xx, indicating that error $xx occurred. Since response processing has not yet been initialized, all commands in the UserInt2.bsp initialization slot call GetBytesCheckError to read ToothPIC responses. During initialization, no responses will be generated by ToothPIC except in response to commands, so calling GetBytesCheckError after each command is acceptable. Once a client is connected, ToothPIC responses might be generated at any time, and simply calling GetBytesCheckError after every command would not work.

After the remaining I/O has been set up, set the Bluetooth device name is set to UserInt2. This is the name which will appear when searching for the device and also in the title bar on the Client. (A version number after the added to the device name for technical reasons; you can remove it once you have finished changing the user interface layout.) The device type is set to Sensor. This governs the icon that is shown when a Client scans for devices. The LEDs are then flashed simultaneously to confirm that initialization has completed OK.

Next, the FlexiPanel server is turned on and the POLLIN / POLLRUN / POLLMODE commands are configured so that when the Data pin goes high, Slot 2 can run. However, the POLLMODE is not activated just yet.

Finally, the variable NextUpdate is set to zero. NextUpdate is used by the main program loop to decide what to do next. Setting it to zero initializes the main program loop so that the first task it will perform is to inspect the value of AN0.

When initialization is complete, control is passed to Slot 1.

Main Program Loop

Slot 1, UI2_MainLoop.bsp, runs whenever the Stamp is not busy performing other tasks. The first thing it does is to release the POLLMODE so that if the Data pin is high, it will be immediately interrupted and the ToothPIC response processed.

If there are no more responses waiting to be processed, it can continue with the tasks it needs to perform. The NextUpdate variable is used to keep a track of what task needs to be performed next, since otherwise it would start from the beginning again each time it was interrupted. A BRANCH instruction then used to perform whichever task is next due.

The eight tasks involve reading the analog inputs AN0 to AN3, and the digital inputs AN4 to AN7. In each case a message is sent to ToothPIC to obtain the input pin value. Note how the POLLMODE is disactivated between sending the message and incrementing NextUpdate. This ensures that it is not interrupted between performing a task and recording that it has performed it. The remainder of the task will be performed in response to the reply received, and takes place in the I/O Response Processing Slot.

Response Processing

Slot 2, `UI2_ProcResponse.bsp`, starts whenever the Data pin goes high indicating there is a response waiting to be processed. Initially, just the first two bytes of the response are read. If it times out during this period, an error message is generated. A timeout error should not occur and if it does so, it is probably an indication that the electrical connections are incorrect.

The first byte indicates the total number of bytes in the message. The second is the 'response byte', which indicates the type of response that the message represents. These bytes are then used to decide how and where to process the response. When the response has been processed, control is passed back to the main program loop.

OK responses

If ToothPIC is given a command for which not specific response is required, it will generate the response byte value $01, which can be interpreted as "OK, command completed successfully". Such responses are frequent and ignored.

Errors and unexpected responses

If the response byte indicates an error, an error message is generated. If the response was not expected, a debug message reports the command to the debug terminal.

I/O and Control responses

If the response byte indicates that the response is in reply to a request for an I/O value, Slot 3 is run. If the response byte indicates that the response is in reply to a request for a user interface control value, or that the user modified a control, slot 4 is run.

Connection, Disconnection and Pairing responses

If the response indicates that a remote device has connected, the red LED is turned on. If the response indicates that a remote device has connected, the red LED is turned off. In this way, it functions as a 'device detected' indicator. If the response indicates that the device connecting or disconnecting is running user interface client software, this information is reported to the debug terminal but no further action is taken.

If the response indicates that a remote device has paired with ToothPIC, this information is reported to the debug terminal but no further action is taken.

Message Box Response

If the response indicates the message box ('Do you wish to reset?') was responded to with a 'Yes' reply, the client is forcefully disconnected and the Stamp resets. Otherwise it continues as normal. Note that only clients from version 3.0 can generate message box responses.

I/O Responses

Slot 3, `UI2_ProcIOVal.bsp`, starts whenever a response is received in reply to a request for an I/O value (see *Main Program Loop*). The length of the response will depend on the I/O value being read: analog values are in the range $000 to $3FF and require two data bytes to be conveyed. Digital values are either 0 or 1 and require only one data byte to be conveyed. Once the data byte(s) have been read, the appropriate control is updated as follows. Controls can be updated whether or not there is actually a client connected.

AN0 Value

The AN0 analog value is logged to the matrix. In a single command, the value is posted to the Y axis and the Z axis value is taken from the real time clock. This provides a simple, powerful data logging facility.

Values ***AN1*** to ***AN3***

The analog values AN1 to AN3 are written to progress-bar-style number controls as a kind of 'Voltmeter' indicator.

Values *AN4* to *AN7*

The digital values AN4 to AN7 are written to text controls. If high, the *ANx* text appears. Otherwise, the text box is cleared.

I/O Responses

Slot 4, `UI2_ProcCtlVal.bsp`, starts whenever a response is received indicating that a control was modified by the user, or in reply to a request for a control value (see later in this section). In either case, bytes three and four of the response will contain the ID value of the control in question.

If the response indicates that a control has been modified, there are no more bytes in the message and the action taken depends on the control.

***Img* image and *SDO* button controls**

If the control was an image control or a button control, the message indicates that the button or image was pressed. If the SDO button was pressed, a short pulse is output on the SDO output pin. You can attach an LED to the SDA pin to see this.

If the image button was pressed, a message box is displayed asking the user whether they wish to reset the Stamp. The response to the message is processed by slot 2, `UI2_ProcResponse.bsp`.

***CCPx* number, *SDA* latch, *DT* date-time and *PA3* list controls**

If the control was any control other than an image or button, it indicates that the control was modified by the user. A message is sent to enquire the new value of the control. When a response to this enquiry is received, it is also handled in slot 4 (see next section).

Control value responses

If the response is in reply to a request for a control value (see previous section), the length of the response will depend on the control type.

Latch controls have binary values, so the response data value is one byte. When the value of the SDA latch control is received, the SDA is set or cleared according to the state of the latch. You can attach an LED to the SDA pin to see this.

List controls have integer values, so the response data value is four bytes. When the value of the PA3 list control is received, all but the lowest byte can be ignored since there are only eight list items. The lowest byte value is used to set the parallel digital outputs on pins AN9 to AN11. You can attach LEDs to the AN9 to AN11 pins to see this.

Number controls, including slider controls, have integer values, so the response data value is four bytes. When the value of the CCPx slider control is received, all but the lowest two byte can be ignored since the allowed range is $000 to $3FF. The lowest two byte values are used to set the PWM duty cycle outputs on pins CCP1 to CCP3. You can attach LEDs to the CCP1 to CCP3 pins to see this – their brightness will be proportionate to the PWM duty cycle.

Designing User Interfaces

☞ *Refer to the DARC-II Firmware Solution documentation available from www.FlexiPanel.com for a graphical overview on how the FlexiPanel Designer software works.*

FlexiPanel User Interfaces are designed with *FlexiPanel Designer.* This is software freely available from *www.FlexiPanel.com*. *FlexiPanel Designer* can also simulate user interfaces on remote client devices.

This section provides an overview of the user interface designs possible with ToothPIC. For full details, consult the *FlexiPanel Designer* software documentation.

Design Sequence

It is also good practice to complete a user interface design as much as possible before coding. This is not for the sake of easier coding; it is because the result is more intuitive to the user.

Control IDs

Beware that control ID value may change if you insert a dialog or a control earlier in FlexiPanel Designer's control list. You may therefore wish to define constants in your host controller code to simplify changes to ID values.

Memory Requirements

FlexiPanel User Interfaces can be written to ToothPIC Stamp Edition at any time using FlexiPanel Designer as shown in the tutorial part VI. The RAM space is limited to `$800` bytes less the number of bytes allocated for the message queue (22 bytes per message). The Flash ROM space is limited to `$E000` bytes. The Flash ROM space is limited to `$E000` bytes.

Programming a User Interface

FlexiPanel User Interfaces can be written to ToothPIC Stamp Edition at any time using FlexiPanel Designer. To program the ToothPIC:

1. Power it up with the pushbutton held down so both LEDs flash

2. Connect to the ToothPIC from your Windows PC using your Bluetooth manager.

3. In `Designer.exe`, check that the Programming COM port 'Target Property' is the same port number that the Bluetooth manager used to connect to the ToothPIC.

4. Select Target Device > Program ToothPIC Stamp Edition, save the computer generated file and then press the Program button to program the ToothPIC.

5. Remember to update any computer generated constants in your BASIC files each time you modify the user interface.

The RAM space is limited to `$800` bytes less the number of bytes allocated for the message queue (22 bytes per message). The Flash ROM space is limited to `$E000` bytes.

FlexiPanel Bluetooth Protocol

FlexiPanel client devices can connect to the *FlexiPanel BASIC Stamp Programmer* (the 'Server') at any time. Once connected, the server tells the client to show the user interface on its display. Both the client and the BASIC Stamp can modify the user interface controls at any time.

The client or ToothPIC may choose to disconnect. Additionally, the link may be dropped if the devices go out of range of each other. The state of the controls is retained by the server so that if the client reconnects, or another client connects, the control panel will be in the same state as it was when it was last modified.

Devices incorporating FlexiPanel Servers must be designed taking into account the possibility of a dropped connection. Specifically, no action should be taken which relies on a client's ability to maintain a connection. If FlexiPanel is used to operate machinery, for example, the ToothPIC should provide a failsafe mode in case the connection is dropped.

The communication standard used by ToothPIC in order to communicate with clients is *FlexiPanel Protocol 3.0*. Some client software may use FlexiPanel Protocol 2.3, which cannot display Image controls.

Introduction to FlexiPanel Controls

A variety of control types are provided by FlexiPanel. These include controls familiar to Windows users and others that are particularly appropriate for FlexiPanel technology.

FlexiPanel clients are required to provide all the requested controls in some form or other. Since the user interface may vary from one FlexiPanel client to another, the appearance may vary.

If the developer expects a device to be used in conjunction with a specific type of FlexiPanel client (e.g. Pocket PC), the appearance on those devices may be specified in more detail from within FlexiPanel Designer.

Some controls are either modifiable or non-modifiable. If a control is non-modifiable, the server may change its value but the client may not.

Dialogs

Controls are arranged in groups called dialogs and ToothPIC can switch between dialogs as required.

Text Control

The text control contains a text string. It will have a fixed maximum length, specified when the control is created.

A text control may have password style, in which case the text entered in the control is not readable by the user.

Button Control

A button control registers when a button is pressed.

Latch Control

A latch control stores a binary (*on/off*) value. Latches may be arranged in groups so that when one latch is turned on the others are turned off.

Password Control

A password control stores a password and has an open and closed state. In the closed state, the user must enter the password to set it to the open state.

In the open state, the password control may be returned to the closed state at any time.

It is possible to specify that password may be modified by the user once the control is in the open state. A master password may also be provided.

Other controls may be directly linked to the password control so that they are only visible when the password is open.

Passwords are limited to 17 Unicode characters or 34 ASCII characters, including zero terminator.

Number Control

A number control stores a numeric value. It is essentially a signed four-byte integer, but its decimal place may be shifted left or right in order to represent any floating-point value.

Matrix Control

A matrix control stores an array of numbers. These might be displayed as a table or a chart. In this release of FlexiPanel, the values are not modifiable by the client.

List Control

A list control allows one item to be selected from a list. The contents of the list may not be modified.

Section Control

A section control acts like a pop-up menu. Controls enclosed within a section control are only visible when the section control is opened.

Controls enclosed inside a closed section control are not transmitted to the client, thereby minimizing communication time.

Section controls predate the dialog facility in ToothPIC. In general, Dialog controls are a more flexible method for managing user interface appearance.

DateTime Control

A DateTime control stores a DateTime value, i.e. second, minute, hour, date, day-of-week, month and year.

The *Real Time Clock* option allows one Date Time control to be updated by the FlexiPanel Programmer's on-board Real Time Clock. To keep the communications burden moderate, the clock is updated only every five seconds. If this control's values are modified by a client or ToothPIC, the Real Time Clock's time is updated accordingly.

Message Control

A message control displays a message on request. If the client is Protocol 3.0 compatible, the message can have a response, e.g. OK or Cancel.

Blob Control

The blob (Binary Large Object) control allows client and server to pass binary objects to each other. It is intended primarily for future expansion and customization. Due to the limitations of some client devices, a client is not obliged to support all features associated with this control; some clients may simply ignore it.

In this release of FlexiPanel, the only use of the blob object is to pass the name of a URL (i.e. web page address) to the client.

Files Control

The files control allows ToothPIC to send files to the Client. The primary use of this feature is to pass HTML files (and related images) so a web browser on the client could display the files. Since the files are stored on ToothPIC, an internet connection is not required.

The files control is intended to allow ToothPIC to upload an instruction manual to the client. In practice, the files might amount to tens or hundreds of kilobytes, so external memory would be required.

Image Control

The image control displays a rectangular image on the client screen. Currently the images are non-modifiable and must be in GIF format. To reduce storage requirements, 16-color GIFs are recommended. Image controls can be made 'clickable' and can be treated as buttons.

Image controls were introduced with FlexiPanel version 3.0. If the client connected is version 2, the image control will not be transmitted. Some clients (such as phones) may not be able to display the image and may just depict it as a button, labeled with the control's name, instead of the image.

Command / Response Reference

The commands and responses detailed here are mostly also generated in a cut-and-pasteable format by FlexiPanel Designer. You should not need to consult this section unless you need to understand the more advanced functions in depth.

Commands

Binary commands may be up to 22 bytes long; ASCII commands 48 bytes. The first byte is the *command length byte*, equal to the total number of bytes in the message. The second byte is the *command byte*, which indicates how the remainder of the message should be interpreted.

Commands can be in either ASCII or binary and the two formats can be mixed freely. Normally you would send binary commands frorm the Stamp. The ASCII commands are intended for evaluation purposes. In ASCII format, each byte is transmitted as two hexadecimal digits (upper or lower case) and the entire command must be followed by a <CR><LF> pair (i.e. the control characters $0D and $0A).

If the *Responses anytime* property is set, all commands generate a response. This will be the `$02 $01` 'OK' response if no other response is appropriate.

Only one command can be processed at once. While it is being processed, the RTS pin will go high and no further messages should be sent. To know when the previous command has completed, observe the state of the RTS pin or wait for a response to be sent. Only then send another command.

Command Summary		
Command	***Command Byte***	***Effect***
Reset	`$00`	Resets ToothPIC
Configure ToothPIC	`$01`	Configures ToothPIC Stamp Edition
Configure I/O	`$02`	Configures I/O
Set I/O	`$03`	Sets an I/O value
Get I/O	`$04`	Requests an I/O value
BlueMatik Command	`$05`	Sends a command to the BlueMatik Bluetooth radio
FlexiPanel Command	`$06`	Sends a command to the FlexiPanel server
User Interface Info	`$07`	Gets user interface information
Get Control Data	`$08`	Gets the value of a control
Set Control Props	`$09`	Sets a control's properties
Set Control Data	`$0A`	Sets the value of a control
Set Row	`$0B`	Sets a row of a matrix control
Append Row	`$0C`	Appends a row of a matrix control
Log Row	`$0D`	Appends a time-stamped row of a matrix control
Read Memory	`$0E`	Reads from memory locations
Write Memory	`$0F`	Writes to memory locations
Set Message	`$40`	Requests the next message (if "messages anytime" mode is not enabled)

Reset Command

The command byte `$00` instructs the ToothPIC to reset. Additionally, a command length byte of zero will generate an immediate reset (and will not wait for <CR><LF> if in ASCII format).

Reset Command Examples	
Reset (binary)	`$02 $00`
Reset (ASCII)	`"0200<CR><LF>"`
Reset (binary)	`$00`

Reset Command Examples	
Reset (ASCII)	"00"

Configure ToothPIC Command

The command byte $01 configures the general properties of the ToothPIC Stamp Edition. The byte after the command byte is the *Property Byte*, which specifies the exact property being set. The remaining bytes represent the new property value, as follows:

Configure ToothPIC Stamp Edition Command Properties		
Property Byte	***Property***	***Remaining Byte(s)***
$01	Security level*	00 = None (default) 01 = Authentication 02 = Authentication and encryption
$02	Baud rate*	01 = 1220 baud 02 = 2400 baud 03 = 4800 baud 04 = 9600 baud (default) 05 = 19200 baud 06 = 38400 baud 07 = 57600 baud 08 = 115200 baud
$03	Authentication PIN*	Zero terminated ASCII pin code (maximum 16 characters plus zero terminator) Default is 0000.
$04	Device name*	Zero terminated ASCII device name (maximum 16 characters plus zero terminator) Default is ToothPIC SE.
$05	Flow control*	00 = None 01 = CTS on INT1, RTS on INT0 02 = CTS on INT1, RTS on SDO 03 = CTS on INT1, RTS on CCP1 04 = CTS on INT1, RTS on AN11 05 = CTS on INT1, RTS on SCL (default)
$06	Host has Rx buffer*	00 = CTS is be strictly observed (default) FF = Host can accept one more byte after CTS goes high
$07	Response queue length*	1 byte = number of responses that can be queued (see notes, default is 32)
$08	Initialization response*	00 = No initialization response (default) FF = Generate an initialization response (default)
$09	Responses anytime*	00 = Transmits responses only in reply to a *GetResponse* command FF = Transmits responses immediately RTS permits it (default)
$0A	ASCII responses*	00 = Generates binary responses (default) FF = Generates ASCII responses
$0B	On internal error…*	01 = Flash error number, reset on button press 02 = Reset immediately 03 = Send error response to host (default)

Configure ToothPIC Stamp Edition Command Properties		
Property Byte	***Property***	***Remaining Byte(s)***
`$0C`	I2C memory setup*	`00` = No I2C memory (default) `01` = I2C memory with 100kHz clock speed `02` = I2C memory with 400kHz clock speed
`$0D`	Device class*	3-bytes A, B, C of the device class value as defined in Device Classes section
`$0E`	INT0 is Data*	`00` = INT0 is not specially configured `FF` = INT0 is high output if messages are queued waiting for a *GetResponse* command or CTS flow control, low otherwise. (default)
`$41`	Daylight Savings	1-byte ***`DSTEvent`*** value as defined in the Daylight Savings Time section (`00` = None, default)
`$42`	Set Date / Time	8-bytes: second byte (0-59), minute byte (0-59), hour byte (0-23), date byte (1-31), day of week byte (0=Sun…6=Sat, 7=calculate from date please), month byte (1-12), year word (0-65535).
`$43`	Request Date / Time	No additional bytes. Generates a Date Time response immediately.

Notes:

Items marked *: Items marked * are stored permanently in EE or Flash memory. With the exception of the PIN code, these changes will not take effect until after the device next resets. If required, the default values can be restored by reloading the ToothPIC Stamp Edition using Wireless Field Programming.

Host has Rx buffer: ToothPIC can operate more efficiently if it can pre-load the next byte into the transmit buffer while the current byte is being transmitted. However, if CTS goes high, it will be too late to stop the byte being sent. Therefore if the host has no receive buffer and cannot accept this last byte after it has placed CTS high, *Host has Rx buffer* should be set to zero. This is required for use with BASIC Stamps.

Response queue length: The number of messages which can be queued up at a time. If the queue fills up before the host can process the messages, a Queue Full Error response will be placed at the top of the queue and it must be assumed that other responses may have been lost. The minimum permitted queue length is 2. The maximum depends on the storage requirements for the FlexiPanel User Interface data, if any. Each response in the queue requires $16 bytes. The total queue byte requirement, plus any RAM data used by the FlexiPanel User Interface, must not exceed $800 bytes. It is up to you to check this; ToothPIC cannot do it for you. If Responses Anytime is disabled, `0201` OK responses will not be added to the queue.

Responses anytime: If you disable Responses Anytime, responses will not be sent unless an `0240` Get Message command is sent.

INT0 is Data: If you use this you should not use INT0 as an RTS pin. The Data pin will be high while there are responses in the queue awaiting an `0240` Get Message. If Responses Anytime is disabled, `0201` OK responses will send the Data pin high.

I2C Memory Setup: If you use this you should not use SCL as an RTS pin. See the section on ToothPIC Memory for more details.

Initialization response: If *Initialization response* is set, the initialization response will be sent irrespective of the *Responses anytime* property.

Configure ToothPIC Stamp Edition Command Examples	
Set authentication security (binary)	`$04 $01 $01 $01`

Configure ToothPIC Stamp Edition Command Examples	
Set authentication security (ASCII)	`“04010101<CR><LF>”`
Set 19200 baud (ASCII)	`“04010205<CR><LF>”`
Set PIN *1234* (ASCII)	`“0801033132333400<CR><LF>”`
Set device name *Fred* (ASCII)	`“0801044672656400<CR><LF>”`
Set binary responses (ASCII)	`“04010A00<CR><LF>”`
Set cellphone device class (ASCII)	`“06010D9FE204<CR><LF>”`
Set time to *13:24:50, April 1st, 2005* (ASCII)	`“0B014232180D010004D507<CR><LF>”`
Get time (ASCII)	`“030143<CR><LF>”`

Configure I/O Command

The command byte `$02` configures the ToothPIC I/O. The byte after the command byte is the *Property Byte*, which specifies the exact I/O property being set. For full details of the I/O electrical capabilities, please refer to the documentation for the PIC18F6720 microcontroller, available from Microchip Technology.

The remaining bytes represent the new property value, as follows:

Configure I/O Command Properties		
Property Byte	***Property***	***Remaining Byte(s)***
`$01`	A to D channels	1 byte, range `00` to `0C` = Number of analog to digital channels (from AN0 up).
`$02`	Negative voltage reference	`00` = Vss is –ve voltage reference (default) `01` = AN2 is –ve voltage reference
`$03`	Positive voltage reference	`00` = Vdd is +ve voltage reference (default) `01` = AN3 is +ve voltage reference
`$04`	PWM time base units	`00` = Turn PWM off `01` = PWM on, base time unit is 0.2µs `02` = PWM on, base time unit is 0.8µs `03` = PWM on, base time unit is 3.2µs
`$05`	PWM period	Range `00` to `FF` = PWM period in PWM base time, less one.
`$06`	Parallel I/O A function	`00` = Not used (default) `02` = 2-bit output (AN11 – AN10) `03` = 3-bit output (AN11 – AN9) `04` = 4-bit output (AN11 – AN8) `05` = 5-bit output (AN11 – AN7) `06` = 6-bit output (AN11 – AN6) `07` = 7-bit output (AN11 – AN5) `12` = 2-bit input (AN11 – AN10) `13` = 3-bit input (AN11 – AN9) `14` = 4-bit input (AN11 – AN8) `15` = 5-bit input (AN11 – AN7) `16` = 6-bit input (AN11 – AN6) `17` = 7-bit input (AN11 – AN5)

Configure I/O Command Properties		
Property Byte	***Property***	***Remaining Byte(s)***
`$07`	Parallel I/O B function	`00` = Not used (default) `02` = 2-bit output (AN4 – AN3) `03` = 3-bit output (AN4 – AN2) `04` = 4-bit output (AN4 – AN1) `05` = 5-bit output (AN4 – AN0) `12` = 2-bit input (AN4 – AN3) `13` = 3-bit input (AN4 – AN2) `14` = 4-bit input (AN4 – AN1) `15` = 5-bit input (AN4 – AN0)
`$10 – $1B`	AN0 – AN11 function (`$10` = AN0, `$11` = AN1, etc)	`00` = Digital input (default) `01` = Digital output (Ignored if configured for A to D input.)
`$1C – $20`	CCP1 – CCP5 function	`00` = Digital input (default) `01` = Digital output `02` = PWM output
`$21 – $22`	INT0 – INT1 function	`00` = Digital input (default) `01` = Digital output
`$23`	SCL function	As INT0
`$24`	SDA function	As INT0
`$25`	SDO function	As INT0

Notes:

I/O pin functions: Pin specifications are ignored for ANx inputs if the *A to D channels* property dictates that they should be A to D inputs. I/O pin functions must not be sent for pins used for parallel I/O, external memory and flow control purposes.

Parallel I/O: Pins are modified or read at exactly the same instant.

PWM base time units: Base time units for PWM values. PWM period is in PWM base time units (1 to 256). PWM duty cycle is in quarter PWM base time units (0 to 1023). *PWM time base units* turns PWM outputs on or off, so during initialization, set up period and initial output values first.

Configure I/O Command Examples	
Set AN0 – AN4 as A to D pins (binary)	`$04 $02 $01 $05`
Set AN0 – AN4 as A to D pins (ASCII)	`“04020105<CR><LF>”`

Set: PWM time base 3.2µs
PWM period as 256 (=1220Hz)
CCP2 pin as PWM (binary)
	`$04 $02 $04 $03`
	`$04 $02 $05 $FF`
	`$04 $02 $1D $02`

Set I/O Command

The command byte `$03` sets a ToothPIC I/O output value. The byte after the command byte is the *Property Byte*, which specifies the exact I/O value being set. The remaining bytes represent the new I/O value, as follows:

Set I/O Command Properties		
Property Byte	***Property***	***Remaining Byte(s)***
`$06`	Parallel I/O A output	Range `00` to `7F` = new value
`$07`	Parallel I/O B output	Range `00` to `1F` = new value
`$10 – $1B`	AN0 – AN11 output	`00` = low, `01` = high

Set I/O Command Properties		
Property Byte	***Property***	***Remaining Byte(s)***
`$1C – $20`	CCP1 – CCP5 output	If digital, as AN0 – AN11. If PWM, range `0000` to `03FF.` (Both bytes must be specified.)
`$21 – $22`	INT0 – INT1 output	As AN0
`$23`	SCL output	As AN0
`$24`	SDA output	As AN0
`$25`	SDO output	As AN0
`$30`	Green LED	`00` = off, `01` = on
`$31`	Red LED	`00` = off, `01` = on

If the pin is not already configured as an output, the value will be ignored. I/O pin value commands must not be sent for pins used for external memory and flow control purposes.

Set I/O Command Examples	
Set AN10 as high (binary)	`$04 $03 $1A $01`
Set AN10 as high (ASCII)	`“04031A01<CR><LF>”`
Set CCP2 as 33% duty cycle (binary)	`$04 $03 $1D $01 $55`

Get I/O Command

The command byte `$04` requests a ToothPIC I/O value. The byte after the command byte is the *Property Byte*, which specifies the exact I/O value being requested. The value will be read and transmitted as an I/O value response.

Get I/O Command Properties	
Property Byte	***Property***
`$06`	Parallel I/O A input
`$07`	Parallel I/O B input
`$10 – $1B`	AN0 – AN11 input
`$1C – $20`	CCP1 – CCP5 input
`$21 – $22`	INT0 – INT1 input
`$23`	SCL input
`$24`	SDA input
`$25`	SDO input
`$30`	Green LED
`$31`	Red LED
`$32`	Pushbutton

If the pin is configured as digital output, the result will be the current output value. If the pin is configured as CCP output, the result will be unpredictable. I/O pin value commands must not be sent for pins used for external memory and flow control purposes.

Get I/O Command Examples	
Get AN10 (binary)	`$03 $04 $1A`
Get AN10 (ASCII)	`“03041A<CR><LF>”`
Get CCP2 (binary)	`$03 $04 $1D`

BlueMatik Command

The command byte `$05` is for the BlueMatik Commands, which are sent directly to the Bluetooth radio. They allow device discovery, connection, transmission of raw data etc. The byte after the command byte is the

Property Byte, which specifies the BlueMatik Property being modified. The remaining bytes represent any data associated with the command.

BlueMatik Command Properties		
Property Byte	***Property***	***Remaining Byte(s)***
`$01`	Enter slave mode	None
`$02`	Connect in master mode	6-byte Bluetooth device ID to connect to
`$03`	Disconnect	None
`$04`	Inquire	1 byte inquiry duration in seconds, range `$02` to `$3C`
`$06`	Enquire link quality	None
`$07`	Enquire signal strength	None
`$08`	Enter sniff mode	None
`$09`	Exit sniff mode	None
`$0A`	Enter hold mode	None
`$0B`	Cancel	None
`$10`	Transmit raw data	Up to 19 bytes of raw data

Notes:

Enter Slave Mode: Device becomes discoverable and connectable. The immediate response will be an *OK* response. When a device connects, a *Connect* Bluematik Response will be generated.

Connect in Master Mode: Device looks for a specific device to connect to. The immediate response will be an *OK* response. When a device connects, a *Connect* Bluematik Response will be generated. If no connect response is generated, the device is not found and you should issue a *Cancel* command.

Cancel: Cancels a Slave, Master or Inquiry command which has not yet completed connecting / inquiry.

For further information: The BlueMatik commands are discussed in detail in the ToothPIC documentation.

User Interface Server Commands

The command byte `$06` is for the User Interface Server Commands, which are sent directly to the FlexiPanel Server. They control the FlexiPanel User Interface service provided by ToothPIC. The user interface must first be programmed from FlexiPanel Designer as described in the above section *Designing User Interfaces*.

BlueMatik commands and FlexiPanel commands cannot be intermixed. BlueMatik must be idle when the FlexiPanel service is started and no further BlueMatik commands may be sent until after the Finish command has been sent.

The byte after the command byte is the *Property Byte*, which specifies the FlexiPanel Server Property being modified. The remaining bytes represent any data associated with the command.

FlexiPanel Server Properties		
Property Byte	***Property***	***Remaining Byte(s)***
`$01`	Start FlexiPanel service	None
`$02`	Finish FlexiPanel service	None
`$03`	Set Dialog	1 byte new dialog index number
`$04`	Disconnect	None
`$09`	Initialize Data	None

Notes:

Set Dialog: Displays a particular dialog. Dialogs are indexed in the order in which they are listed in FlexiPanel Designer, starting from `$00`.

Disconnect: Disconnects a particular client but does not end FlexiPanel service. Another client may connect.

Initialize Data: Data will already be initialized when the FlexiPanel service starts. This command allows the initial settings to be restored.

User Interface Info Commands

The command byte `$07` is for the User Interface Info command, which allows the host to read information about the user interface which has been programmed into ToothPIC by FlexiPanel Designer. The user interface must first be programmed from FlexiPanel Designer as described in the above section *Designing User Interfaces*.

The byte after the command byte is the *Property Byte*, which specifies the User Interface Info Property being modified. The remaining bytes represent any data associated with the command. Each general and dialog info commands will receive one User Interface Info Response; control info commands will receive two. Note that since the ToothPIC Stamp Edition does not know in advance what the user interface looks like, if you ask a meaningless question you may get a meaningless response rather than an error.

User Interface Info Properties		
Property Byte	***Property***	***Remaining Byte(s)***
`$01`	General Info	None
`$02`	Dialog Info	Dialog index
`$03`	Control Info	Control ID

Note: The bytes after the command byte are the dialog / control ID. Beware that this ID may change if you insert a dialog or a control earlier in the control list. You should therefore use the `Designer`-generated constants simplify changes to ID values.

Get Control Data Command

The Get Control Data Command byte `$08` retrieves the value of a control. The user interface must first be programmed from FlexiPanel Designer as described in the above section *Designing User Interfaces*. The FlexiPanel Server does not need to be running in order to get or set values.

The two bytes after the command byte are the Control ID. This is the value displayed in FlexiPanel Designer under heading ID in the main controls list. Beware that this ID may change if you insert a dialog or a control earlier in the control list. You should therefore use the `Designer`-generated constants simplify changes to ID values.

Each Get Control Data Command will receive at least one Control Data Response. If the storage required for the control is greater than 16 bytes, multiple Control Data Response bytes will be received.

Get Control Data Command Examples	
Get control `$1234` (binary)	`$04 $08 $12 $34`
Get control `$1234` (ASCII)	`“04081234<CR><LF>”`

Set Control Props Command

The Set Control Props Command byte `$09` sends control properties information to the client. Properties are modified on the client only, not in the server. A client must therefore be connected for these properties to take effect. When a new client connects, the properties will have to be sent again.

The entire command is always 12 bytes long. The two bytes after the command byte are the Control ID. This is the value displayed in FlexiPanel Designer under heading ID in the main controls list. Beware that this ID may change if you insert a dialog or a control earlier in the control list. You should therefore use the `Designer`-generated constants simplify changes to ID values.

The subsequent 4 bytes of the command specify the new properties. The following properties may be bitwise-ORed together:

Properties of Message Controls *(A message can only be displayed if its dialog is visible.)*	
To display a message control	`$00 $00 $00 $00`
To modify the message control's properties without displaying it.	`$01 $00 $00 $00`
No icon	`$00 $00 $01 $00`
Stop icon	`$00 $00 $02 $00`
Exclamation icon	`$00 $00 $03 $00`
Question icon	`$00 $00 $04 $00`
Information icon	`$00 $00 $05 $00`
No button response message required	`$00 $00 $00 $00`
OK button (V3 clients only)	`$00 $00 $10 $00`
OK, Cancel buttons (V3 clients only)	`$00 $00 $20 $00`
Retry, Cancel buttons (V3 clients only)	`$00 $00 $30 $00`
Yes, No buttons (V3 clients only)	`$00 $00 $40 $00`
Yes, No, Cancel buttons (V3 clients only)	`$00 $00 $50 $00`
Abort, Retry, Ignore buttons (V3 clients only)	`$00 $00 $60 $00`

Properties of Other Controls	
Ensure control is visible on client screen	`$10 $00 $00 $00`
If the control color properties are to be set	`$40 $00 $00 $00`

The final 4 bytes of the command specify the new control color. The RGB values, each in the range `$00` to `$FF`, are sent as the bytes `$RR $GG $BB $00`.

Set Control Properties Command Examples	
Show a message control `$0B03`, info icon, Yes / No buttons (binary)	`$0C $09 $0B $03 $00 $45 $00 $00 $00 $00 $00 $00`
Make control `$0003` onscreen and red.	`$0C $09 $00 $03 $50 $00 $00 $00 $FF $00 $00 $00`

Set Control Data Command

The Set Control Data Command byte `$0A` sets the value of a control. The user interface must first be programmed from FlexiPanel Designer as described in the above section *Designing User Interfaces*. The FlexiPanel Server does not need to be running in order to get or set values. For matrix controls, one of the Set / Append / Log Row commands is preferred.

The two bytes after the command byte are the Control ID. This is the value displayed in FlexiPanel Designer under heading ID in the main controls list. Beware that this ID may change if you insert a dialog or a control earlier in the control list. You should therefore use the `Designer`-generated constants simplify changes to ID values.

The next two bytes are the control offset (little-endian) and the remaining bytes are the new control data. For controls whose data length is 16 bytes or less, the control offset should be zero. For controls whose length is 17 bytes or greater, it will be necessary to update the data using multiple Set Control Data Commands, each with different control offset values. The control offset indicates where the new control data are to be placed within the control's data field. If multiple Set Control Data Commands are sent for a control, ensure that the last, and only the last, has a control offset of zero. This indicates that the control data update is complete and the new data can be transmitted to the client.

The data types for the various controls are as follows:

Control Data Types	
Control	*Data type description.*
Blob	Binary data object – size specified in FlexiPanel Designer.
Button	No data.
Date-Time	8-bytes: second byte (0-59), minute byte (0-59), hour byte (0-23), date byte (1-31), day of week byte (0=Sun…6=Sat, 7=calculate from date please), month byte (1-12), year word (0-65535).
Files	No data.
Image	No data.
Latch	1 byte, `$00` for off, `$FF` for on.
List	4 bytes, little-endian, zero-based index of the selected item.
Matrix	Extensive – see ToothPIC documentation or use set row / log row / append row.
Message	No data.
Number	4 bytes, little-endian, signed integer.
Password	1 byte, `$00` for closed, `$FF` for open.
Section	1 byte, `$00` for closed, `$FF` for open.
Text	ASCII or Unicode text – size specified in FlexiPanel Designer.

Set Control Data Command Examples	
Set list or number control `$0002` to the value `$00000005`	`$0A $0A $00 $02 $00 $00 $05 $00 $00 $00`

Set text control `$0705` to the value "ABCDEFGHIJKLMNOPQRSTUVWXYZ" Note zero terminator and little-endian offset value.
 	
First 16 bytes sent last to ensure that all control data are updated on the client simultaneously.
	`$11 $0A $07 $05 $10 $00 $51 $52 $53 $54 $55 $56 $57 $58 $59 $5A $00`
	
	`$16 $0A $07 $05 $00 $00 $41 $42 $43 $44 $45 $46 $47 $48 $49 $4A $4B $4C $4D $4E $4F $50`

Set Row Command

The Set Row Data Command byte `$0B` sets a single row of data on a matrix control. The user interface must first be programmed from FlexiPanel Designer as described in the above section *Designing User Interfaces*. The FlexiPanel Server does not need to be running in order to get or set values.

The two bytes after the command byte are the matrix Control ID. This is the value displayed in FlexiPanel Designer under heading ID in the main controls list. Beware that this ID may change if you insert a dialog or a control earlier in the control list. You should therefore use the `Designer`-generated constants simplify changes to ID values.

The next two bytes are the row number to be set (little-endian). The first row is row zero. No row should be set unless all row numbers preceding have been previously set. The Set Row Command should not be given on any matrix which has previously received an Append Row or Log Row command.

If the matrix is a Date-Time matrix, the next 8 bytes are the date-time axis data; otherwise, if it is an XY matrix with *n* bytes per X-axis value, the next *n* bytes are the X-axis data.

The remaining rows are the data for each cell in the row. The number of bytes of data should equal the number of bytes per cell multiplied by the number of columns.

Unlike the Append Row and Log Row commands, it is quite probable that you will wish to update more than one row at a time, and only transmit the result to the client when all rows have been updated. For this reason, the client is not updated unless a Set Row command is sent containing no X or Y or row index data (i.e. just the first four bytes of the command).

Set Row Command Example (X Axis is 2 bytes, 3 columns of 1 byte each)	
Set matrix `$0002` row 6 to (X=33, 6, 7, 8)	`$0B $0B $00 $02 $06 $00 $21 $00 $06 $07 $08`
Set matrix `$0002` row 4 to (X=22, 3, 4, 5)	`$0B $0B $00 $02 $04 $00 $16 $00 $06 $07 $08`
Send result to client	`$04 $0B $00 $02`

Append Row Command

The Append Row Data Command byte `$0C` appends a row onto the end of a matrix control. The user interface must first be programmed from FlexiPanel Designer as described in the above section *Designing User Interfaces*. The FlexiPanel Server does not need to be running in order to get or set values.

The two bytes after the command byte are the matrix Control ID. This is the value displayed in FlexiPanel Designer under heading ID in the main controls list. Beware that this ID may change if you insert a dialog or a control earlier in the control list. You should therefore use the `Designer`-generated constants simplify changes to ID values.

The row data will be appended onto the bottom of the matrix. If the matrix is full, the top row will be discarded.

If the matrix is a Date-Time matrix, the next 8 bytes are the date-time axis data; otherwise, if it is an XY matrix with *n* bytes per X-axis value, the next *n* bytes are the X-axis data.

The remaining rows are the data for each cell in the row. The number of bytes of data should equal the number of bytes per cell multiplied by the number of columns.

The updated row will be immediately transmitted to the client, if connected.

Append Row Command Example (X Axis is 2 bytes, 3 columns of 1 byte each)	
Append (X=33, 6, 7, 8) to matrix `$0002`	`$09 $0C $00 $02 $21 $00 $06 $07 $08`

Log Row Command

The Log Row Data Command byte `$0D` appends a row onto the end of a Date-Time matrix control, using the current value of the real time clock as the Date-Time axis data. The user interface must first be programmed from FlexiPanel Designer as described in the above section *Designing User Interfaces*. The FlexiPanel Server does not need to be running in order to get or set values.

The two bytes after the command byte are the matrix Control ID. This is the value displayed in FlexiPanel Designer under heading ID in the main controls list. Beware that this ID may change if you insert a dialog or a control earlier in the control list. You should therefore use the `Designer`-generated constants simplify changes to ID values.

The row data will be appended onto the bottom of the matrix. If the matrix is full, the top row will be discarded.

The remaining rows are the data for each cell in the row. The number of bytes of data should equal the number of bytes per cell multiplied by the number of columns.

The updated row will be immediately transmitted to the client, if connected.

Log Row Command Example (X Axis is date-time, 3 columns of 1 byte each)	
Append (6, 7, 8) to matrix `$0002`	`$07 $0D $00 $02 $06 $07 $08`

Get Memory Command

The Get Memory Data command byte `$0E` retrieves data from internal or external memory. The byte after the command byte is the `mStr` memory storage type as defined in the *ToothPIC Memory* section. The next two bytes are the `Addr` memory address (little-endian) as defined in the *ToothPIC Memory* section.

The final byte is the number of bytes required, up to 17 bytes.

Each Get Memory Command will receive exactly one Got Memory Response.

Get Control Data Command Example	
Get 4 bytes of EE memory from `$0123`	`$06 $0E $03 $23 $01 $04`

Set Memory Command

The Set Memory Data command byte `$0F` sets data in internal or external memory. Very little error checking is done with this command, so it should be used with care. It is your responsibility to ensure that you do not overwrite important memory regions. Overwriting `ROM00` locations below `$C000` may prevent Wireless Field Programming from functioning.

The byte after the command byte is the `mStr` memory storage type as defined in the *ToothPIC Memory* section. The next two bytes are the `Addr` memory address (little-endian) as defined in the *ToothPIC Memory* section.

The remaining bytes are the data to be written to memory `mStr` starting at location `Addr`. Up to 17 bytes may be written per Set Memory command.

Set Memory Command Example	
Set EE memory from `$0123` with the values `$05 $06 $07 $08`.	`$09 $0F $03 $23 $01 $05 $06 $07 $08`

The following regions of memory are not used and you may use them as you wish. Note that the actual location of RAM in the PIC18F6720 is from $0100; the actual location of ROM is from $010000.

Type	***mStr value***	***Addr min***	***Addr max***
RAM	`$02`	`$0000`	`$08FF`
EE	`$03`	`$0000`	`$3DF`
External memory I2C address `$B0` / `$B1`	`$10`	`$0000`	IC limit

Type	*mStr value*	*Addr min*	*Addr max*
External memory I2C address $B2 / $B3	`$11`	`$0000`	IC limit
External memory I2C address $B4 / $B5	`$12`	`$0000`	IC limit
External memory I2C address $B6 / $B7	`$13`	`$0000`	IC limit
External memory I2C address $B8 / $B9	`$14`	`$0000`	IC limit
External memory I2C address $BA / $BB	`$15`	`$0000`	IC limit
External memory I2C address $BC / $BD	`$16`	`$0000`	IC limit
External memory I2C address $BE / $BF	`$17`	`$0000`	IC limit
ROM (Flash)	`$81`	`$0000`	`$FDC0`

Breakpoint Command

The command byte `$10` instructs the ToothPIC to flash its LEDs, allowing the host to indicate that an error has occurred. It there is no response and no further commands should be sent without resetting the ToothPIC.

The byte after the command byte is a `FlashVal` value. This value is used to control how the LEDs flash. The number is flashed as follows:

- As many green and red simultaneous flashes as the hundreds digit of `FlashVal`
- As many red flashes as the tens digit of `FlashVal`
- As many green flashes as the units digit of `FlashVal`

or

- Three green and red simultaneous flashes if `FlashVal` is zero

Breakpoint Command Examples	
Flash 201 (binary)	`$03 $10 $C9`

Get Message Command

The command byte `$40` instructs the ToothPIC to send the next response in the queue, presuming the *Responses anytime* property is not set. If there are no more responses, the OK response will be sent.

Get Message Command Examples	
Get Message (binary)	`$02 $40`
Get Message (ASCII)	`“0240<CR><LF>”`

Responses

Binary responses may be up to 22 bytes long; ASCII responses 48 bytes. The first byte is the *response length byte*, equal to the total number of bytes in the message. The second byte is the *response byte*, which indicates how the remainder of the message should be interpreted.

Responses may be in either ASCII or binary as specified by the *ASCII responses* property. In ASCII format, each byte is transmitted as two hexadecimal digits and the entire command will be preceded and followed by a <CR><LF> pair (i.e. the control characters $0D and $0A).

If the *Responses anytime* property is set, all commands generate a response. This will be the OK response if no other response is appropriate.

Command Summary		
Response	***Response Byte***	***Interpretation***
OK	`$01`	Acknowledges completion of previous command
Error	`$02`	Reports an error
Date Time	`$03`	Reports the time and date
I/O Value	`$04`	The value of the requested I/O pin
BlueMatik Response	`$05`	The BlueMatik radio reports that an event occurred
FlexiPanel Response	`$06`	The FlexiPanel server reports that an event occurs
User Info	`$07`	The value of the requested user interface information
Control Data	`$08`	The value of the requested control
Got Memory	`$09`	The value of the requested memory location
Initialization	`$53`	A message sent to indicate initialization is complete

OK Response

The response byte `$01` indicates that the previous command has been processed and another command may be sent.

OK Response Examples	
OK (binary)	`$02 $01`
OK (ASCII)	`“<CR><LF>0201<CR><LF>”`

Error Response

The response byte `$02` indicates that an error has occurred. The byte after the response byte is the *Error Byte*, which specifies the exact error that occurred. See the section Error Codes for a full list of possible errors.

Date Time Response

The response byte `$03` reports the current date and time to the host. The 8 bytes after the response byte are the date and time: second byte (0-59), minute byte (0-59), hour byte (0-23), date byte (1-31), day of week byte (0=Sun…6=Sat), month byte (1-12), year word (0-65535)..

Date Time Response Examples – 13:24:50, Friday, April 1st, 2005	
Date Time (binary)	`$0A $03 $32 $18 $0D $01 $05 $04 $D5 $07`
Date Time (ASCII)	`“<CR><LF>0A0332180D010504D507<CR><LF>”`

I/O Value Response

The response byte `$04` reports an I/O input value to the host. The first byte after the response byte indicates which value is being reported and matches those used in the Get I/O Command. The remaining byte(s) will contain the reported value. The size will vary according to data type as shown in the examples below.

I/O Value Response Examples	
SDO digital input high (binary)	`$04 $04 $25 $01`
SDO digital input low (ASCII)	`“<CR><LF>04042500<CR><LF>”`
AN3 analog input $234 (range 0 to 1023) (ASCII)	`“<CR><LF>0504130234<CR><LF>”`
Parallel A (7-bit) analog input $14 (range 0 to 31) (ASCII)	`“<CR><LF>04040614<CR><LF>”`

BlueMatik Response

The response byte `$05` reports a response from the BlueMatik radio. The first byte after the response byte is the *Response Value* that indicates which response is being reported. The remaining byte(s) will contain any associated data.

BlueMatik Responses			
Response Value	***Response Meaning***	***Associated Data***	***Interpretation***
`$03`	Connect	6-byte Bluetooth ID of remote device	A connection has been established.
`$04`	Disconnect	None	The remote device terminated the connection
`$05`	Found	6-byte Bluetooth ID of remote device, then first 12 bytes of name and zero terminator	A remote device was discovered during inquiry. (Ignore any data send after name zero terminator.)
`$06`	Paired	6-byte Bluetooth ID of remote device	A remote device successfully paired with this device
`$07`	LinkQ	1-byte link quality	Response to link quality request
`$08`	Signal	1-byte signal strength	Response to signal strength request
`$09`	Low Power	None	The low power mode changed
`$10`	Receive raw data	Up to 19 bytes	Raw data received

For more details on any of these responses, consult the BlueMatik Events section of the main ToothPIC documentation.

BlueMatik Response Examples	
Device 12:34:56:78:90:AB connected (ASCII)	`“<CR><LF>0905031234567890AB<CR><LF>”`
Link quality (Binary)	`$04 $05 $07 $FF`

FlexiPanel Server Response

The response byte `$06` reports a response from the FlexiPanel Server. The first byte after the response byte is the *Response Value* that indicates which response is being reported. The remaining byte(s) will contain any associated data.

FlexiPanel Server Responses			
Response Value	***Response Meaning***	***Associated Data***	***Interpretation***
`$01`	Connect	None	A client connected
`$02`	Disconnect	None	A client disconnected
`$04`	Client Update	2-byte control ID	Client modified a control
`$07`	Message Response	1-byte index number of response	The client selected an option in a message box. (V3 clients only)

For the Client Update message, the Control ID is the value displayed in FlexiPanel Designer under heading ID in the main controls list. Beware that this ID may change if you insert a dialog or a control earlier in the control list. You may therefore wish to define constants in your host controller code to simplify changes to ID values.

For more details on any of these responses, consult the FlexiPanel User Interface Server Events section of the main ToothPIC documentation section.

FlexiPanel Server Response Examples	
Device connected (ASCII)	`“<CR><LF>030601<CR><LF>”`
Client modified control `$0002` (Binary)	`$05 $06 $04 $00 $02`

User Interface Info Response

The response byte `$07` is a response to a User Interface Info command. The first byte after the response byte is the *Response Value* that indicates which response is being reported. The remaining byte(s) will contain associated data.

User Interface Info Responses		
Response Value	***Response Meaning***	***Associated Data*** *(2-byte integers except control ID are are little-endian.)*
`$01`	General Info	First 16 bytes of `bqFxPData` data structure as described in `ToothPIC.h`
`$02`	Dialog Info	First byte is Dialog ID; then next 4 bytes are `bqFxPDlgData` data structure as described in `ToothPIC.h`
`$03`	Control Info I	First byte is Control ID; then next 4 bytes are `bqFxPDlgData` data structure as described in `ToothPIC.h`
`$04`	Control Info II	First byte is Control ID; then next 4 bytes are `bqFxPDlgData` data structure as described in `ToothPIC.h`

For the Client Update message, the Control ID is the value displayed in FlexiPanel Designer under the heading ID in the main controls list. Note that this ID may change if you insert a dialog or a control earlier in the control list. You may therefore wish to define constants in your host controller code to simplify changes to ID values.

Control Data Response

Control Data Response byte `$08` is a response to a Get Control Data command. The two bytes after the response byte are the Control ID.

If the control value comprises more than 16 bytes, it will be necessary to send multiple control data response messages to convey the information. The two bytes after the Control ID are therefore the data offset (little-endian) of this particular response: `$0000` for the first response, `$0010` for the second and so on. The remaining byte(s) will contain associated data.

For the Client Update message, the Control ID is the value displayed in FlexiPanel Designer under the heading ID in the main controls list. Note that this ID may change if you insert a dialog or a control earlier in the control list. You may therefore wish to define constants in your host controller code to simplify changes to ID values.

The data types for the various controls are as follows:

Control Data Types	
Control	*Data type description.*
Blob	Binary data object – size specified in FlexiPanel Designer.
Button	No data.
Date-Time	8-bytes: second byte (0-59), minute byte (0-59), hour byte (0-23), date byte (1-31), day of week byte (0=Sun…6=Sat), month byte (1-12), year word (0-65535).
Files	No data.
Image	No data.
Latch	1 byte, `$00` for off, `$FF` for on.
List	4 bytes, little-endian, zero-based index of the selected item.
Matrix	Extensive – see ToothPIC documentation.
Message	No data.
Number	4 bytes, little-endian, signed integer.
Password	1 byte, `$00` for closed, `$FF` for open.
Section	1 byte, `$00` for closed, `$FF` for open.
Text	ASCII or Unicode text – size specified in FlexiPanel Designer.

For more details on any of these responses, consult the BlueMatik Events section of the ToothPIC documentation.

Got Memory Response

Control Data Response byte `$09` is a response to a Get Memory Data command.

The byte after the command byte is the `mStr` memory storage type as defined in the *ToothPIC Memory* section. The next two bytes are the `Addr` memory address (little-endian) as defined in the Memory *ToothPIC Memory* section.

The remaining bytes are the data read from memory `mStr` starting at location `Addr`, as requested

Got Memory Response Example	
Report EE memory from `$0123` being values `$05 $06 $07 $08`.	`$09 $09 $03 $23 $01 $05 $06 $07 $08`

Initialization Response

The initialization response byte $53 indicates that the ToothPIC Stamp Edition has initialized correctly. The 16 bytes of the response including the response length byte will be comprised of the byte $0B the ASCII characters for "ToothPIC SE". Note the initialization response is disabled by default.

Initialization Response Examples – Version 3.0.00002	
Initialization complete (binary)	$0B then "ToothPIC SE"

Command / Response User Guide

The following tables of groups of commands and responses show how to achieve common tasks with ToothPIC Stamp Edition. The responses are examples and may differ.

Initialization	
Response	*Meaning*
$0B then "ToothPIC SE"	Initialization successful

Setting SDO as digital output		
Command	Response	Meaning
04022501	0201	Set SDO to output
04032501	0201	Set output value to high

Setting SDO as digital input		
Command	*Response*	*Meaning*
04022300	0201	Set SCL to input
030423	04042300	Get input value, result $00 = low

Setting CCP1 as CCP output		
Command	*Response*	*Meaning*
04020403	0201	Set PWM time base to 3.2µs
040205FF	0201	Set PWM period to ($FF+1) x 3.2µs = 819.2µs (1220Hz)
04021C02	0201	Set CCP1 to PWM output
05031C01FF	0201	Set duty cycle to $01FF x (3.2µs / 4) = 408.8µs

Setting AN11 as analog input		
Command	*Response*	*Meaning*
04020102	0201	Set AN0 and AN1 as analog inputs
030411	05041103EB	Read AN1

Setting AN11, AN10, AN9 as parallel output		
Command	*Response*	*Meaning*
04020603	0201	Set parallel output A to 3-bit
04030605	0201	Set output to $05 = 101 binary

Setting AN11, AN10, AN9 as parallel input		
Command	*Response*	*Meaning*
04020613	0201	Set parallel input A to 3-bit
030406	04040603	Read input value, result $03 = 011 binary

Bluetooth slave mode connection		
Command	*Response*	*Meaning*
030501	0201	Enters slave mode
	090503080046B939B0	Remote device with ID 80:00:46:B9:39:B0 connects
	08051048656C6C6F	†Remote device sent ASCII Hello
	030504	†Remote device disconnected

† Note that the master / slave distinction merely refers to which device establishes the connection. Both devices may send data to the other. Either device may choose to disconnect; in addition, automatic disconnection will occur if they go out of range of each other. BlueMatik commands generating an OK response (0201) should not be regarded as complete until the OK response is generated.

Bluetooth inquiry & master mode connection		
Command	*Response*	*Meaning*
0405040F	(Busy - No immediate response)	Do device inquiry for $0F seconds
	160505080046B939B0466C 65786950616E656C205600	Remote device with ID 80:00:46:B9:39:B0 found, name begins "FlexiPanel"
	16050500043E80C44B506F 636B65745F504300656C00	Remote device with ID 80:00:46:B9:39:B0 found, name begins "Pocket_PC"
	030504	End of device inquiry (command complete)
0905028000 46B939B0	090502800046B939B0	Connect to remote device with ID 00:04:3E:80:C4:4B; connection successful
0805104865 6C6C6F	0201	†Send ASCII "Hello" to remote device.
030506	0201 030507FF	Enquire link quality Quality is reasonably good
030503	030504 0201	†Disconnect from remote device

FlexiPanel Server (results depend on UI loaded)		
Command	*Response*	*Meaning*
030601	0201	Start FlexiPanel UI Service
	09050300043E80C44B	BlueMatik reports remote device connected.
	030601	FlexiPanel Service reports FlexiPanel Client connected. (Choose Load Recommended Layout from menu if layout unusual.)
04060301	0201	Show dialog $01
	0506040109	Control ID = $0109 modified
030602	0201 030504	End FlexiPanel service. BlueMatik reports client disconnecting

User Interface Info (results depend on UI loaded)		
Command	*Response*	*Meaning*
030701	1307011002AA0006000 ABA6000450888420000	User interface general information
04070202	0707020D0004BA	User interface dialog $02 information
0507030204	130703000100FF4001F FFF000100FF10013A01 0F0704000100FF00011 900000100FF	User interface control $0204 information Response part I Response part II

Control Data Responses (results depend on UI loaded)		
Command	*Response*	*Meaning*
`04080102`	`0E080102000000` `36160D020CD407`	Fixed date-time control `$0102` initial value (8 bytes of data)

`04080101`	`16080101000054657374 20` `5265616C2054696D652043`
	
	 `0B08010110006C6F636B00`
		Control `$0101` (“Test real time clock”) text; first 16 bytes of ASCII text
		
		 Remaining bytes of ASCII text

ToothPIC Memory

Memory Map

The following memory areas are accessible:

Type	*mStr value*	*Lower address limit*	*Upper address limit*	*Also used by*
RAM (volatile)	`0x02`	`0x100`	`0x8FF`	User interface (from `$100` up). Message queue (from `$8FF` down).
EE	`0x03`	`0x000`	`0x3FF`	User interface (from `$0000` up). Configuration settings `$3E0-$3FF`
EXT0 external memory (I2C addr `0xB0/0xB1`)	`0x10`	`0x0000`	IC limit	User interface (from `$0000` up).
EXT1 external memory (I2C addr `0xB2/0xB3`)	`0x11`	`0x0000`	IC limit	User interface (from `$0000` up).
EXT2 external memory (I2C addr `0xB4/0xB5`)	`0x12`	`0x0000`	IC limit	User interface (from `$0000` up).
EXT3 external memory (I2C addr `0xB6/0xB7`)	`0x13`	`0x0000`	IC limit	User interface (from `$0000` up).
EXT4 external memory (I2C addr `0xB8/0xB9`)	`0x14`	`0x0000`	IC limit	User interface (from `$0000` up).
EXT5 external memory (I2C addr `0xBA/0xBB`)	`0x15`	`0x0000`	IC limit	User interface (from `$0000` up).
EXT6 external memory (I2C addr `0xBC/0xBD`)	`0x16`	`0x0000`	IC limit	User interface (from `$0000` up).
EXT7 external memory (I2C addr `0xBE/0xBF`)	`0x17`	`0x0000`	IC limit	User interface (from `$0000` up).
Upper ROM block (`$01xxxx`)	`0x81`	`0x0000`	`0xDFFF`	User interface (from `$0000` up).

This memory is free for you to use, other than as follows:

- `Designer.exe` requires memory for the user interface as specified at the beginning of the computer-generated file it creates.

- RAM locations from `$8FF` downwards are used by the message queue, `$16` bytes per message. For example, by default, `$20` messages can be queued, so `$640` to `$8FF` are used for the message queue.

- EE locations `$3E0` to `$3FF` are used to store configuration settings.

If your user interface uses a large quantity of EE or RAM memory, you must also take care that you do not overwrite the message queue or the configuration settings. FlexiPanel Designer down not know these are already allocated and it will not warn you about this.

You can inspect values outside the address limits specified. However, writing to these addresses may cause the ToothPIC to malfunction.

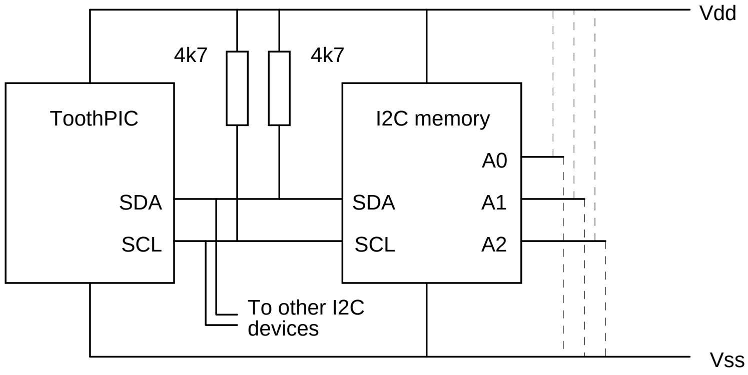
Cautions using memory

If your user interface uses a large quantity of EE or RAM memory, you must also take care that you do not overwrite the message queue or the configuration settings. FlexiPanel Designer down not know these are already allocated and it will not warn you about this.

Writing to Flash ROM memory causes the PIC CPU clock to be suspended for about 2ms. During this period, any incoming ASCII characters may be lost if the UART interrupt cannot be responded to. Avoid using ROM for data modifiable by the FlexiPanel Client unless the UART speed is 2400 baud or less. RAM, EE and External memory have no such limitations.

External memory

Up to eight external I2C memory devices may be placed on the SDA and SCL lines and accessed automatically using the Get Memory and Set Memory commands. External memory access has been tested with Microchip Technology 24Cxxx series EEPROM devices and external I2C memory devices should use the same I2C communications format. Setting up external I2C memory is as follows, with reference to the figure:



1. Send a Configure ToothPIC Flow Control command to change move the RTS signal from SCL to another pin.

2. Send a Configure ToothPIC I2C Memory Setup command to suit your memory, either 100kHz or 400 kHz.

3. Connect the Vdd, Vss , SDA and SCL lines for all memory devices.

4. Provide 4K7 pullup resistors for SDA and SCL.

5. Hardwire each I2C memory device address line A0-A2 to Vcc or Vss to specify separate storage addresses as in the following table.

Storage address	***A2***	***A1***	***A0***
`STR_EXT0`	`Vss`	`Vss`	`Vss`
`STR_EXT1`	`Vss`	`Vss`	`Vdd`
`STR_EXT2`	`Vss`	`Vdd`	`Vss`
`STR_EXT3`	`Vss`	`Vdd`	`Vdd`
`STR_EXT4`	`Vdd`	`Vss`	`Vss`
`STR_EXT5`	`Vdd`	`Vss`	`Vdd`
`STR_EXT6`	`Vdd`	`Vdd`	`Vss`
`STR_EXT7`	`Vdd`	`Vdd`	`Vdd`

6. Add the I2C memory initialization code `I2CMemInit100kHz` or `I2CMemInit400kHz` to your initialization code. `I2CMemInit400kHz` is faster but limited to devices which can operate at this speed.

7. If `Designer.exe` stores user interface information in the external memory, the memory must be connected when the user interface is programmed into the ToothPIC.

8. Send Get Memory and Set Memory commands as required.

Daylight Savings Time

Daylight Savings Time (DST) adjustments may be implemented using the Configure ToothPIC – Daylight Savings Time command, specifying a `DSTEvent` value. When the event related to the `DSTEvent` occurs, the clock is changed and the `DSTEvent` event is exchanged for its seasonal converse.

The `DSTEvent` values are defined in the following table. Please note that Daylight Savings Time data for some countries may become out of date as laws change. We rely on customer feedback to keep this information up to date. Countries in the tropics do not implement Daylight Savings Time. If a specific day of the week is specified, the event is triggered on the first corresponding day on or after the date specified. *We have put considerable effort into compiling this data and we disclose it for customer reference only. It remains our copyright and its reproduction or use without our prior consent is not permitted.*

***DSTEvent* Value**	***Interpretation***	***Month***	***Date***	***Day of week***	***Hour***	***Action taken***
0	No DST (default)	n/a	n/a	n/a	n/a	None
1	Australia winter	10	25	Sunday	0	Hour advanced 1
33	Australia summer	3	25	Sunday	1	Hour retarded by 1
2	Bahamas winter	4	1	Sunday	0	Hour advanced 1
34	Bahamas summer	10	25	Sunday	1	Hour retarded by 1
3	Brazil winter	11	1	Sunday	0	Hour advanced 1
35	Brazil summer	2	15	Sunday	1	Hour retarded by 1
4	Canada winter	4	1	Sunday	0	Hour advanced 1
36	Canada summer	10	25	Sunday	1	Hour retarded by 1
5	Chile winter	10	8	Saturday	0	Hour advanced 1
37	Chile summer	3	8	Saturday	1	Hour retarded by 1
6	Cuba winter	4	1	Any	0	Hour advanced 1
38	Cuba summer	10	25	Sunday	1	Hour retarded by 1
7	Eastern Europe winter	3	25	Sunday	0	Hour advanced 1
39	Eastern Europe summer	10	25	Sunday	1	Hour retarded by 1
8	Egypt winter	4	24	Friday	0	Hour advanced 1
40	Egypt summer	9	24	Thursday	1	Hour retarded by 1
9	Falklands winter	9	8	Sunday	0	Hour advanced 1
41	Falklands summer	4	6	Sunday	1	Hour retarded by 1
10	Greenland winter	3	25	Sunday	1	Hour advanced 1
42	Greenland summer	10	25	Sunday	2	Hour retarded by 1
11	Iran winter	1	1	Any	0	Hour advanced 1
43	Iran summer	7	1	Any	1	Hour retarded by 1
12	Iraq winter	4	1	Any	0	Hour advanced 1
44	Iraq summer	10	1	Any	1	Hour retarded by 1
13	Israel winter	4	1	Friday	0	Hour advanced 1
45	Israel summer	9	1	Friday	1	Hour retarded by 1
14	Kirgistan winter	3	25	Sunday	0	Hour advanced 1
46	Kirgistan summer	10	25	Sunday	1	Hour retarded by 1
15	Lebanon winter	3	25	Sunday	0	Hour advanced 1
47	Lebanon summer	10	25	Sunday	1	Hour retarded by 1
16	Mexico winter	4	1	Sunday	0	Hour advanced 1
48	Mexico summer	10	25	Sunday	1	Hour retarded by 1
17	Namibia winter	9	1	Sunday	0	Hour advanced 1
49	Namibia summer	4	1	Sunday	1	Hour retarded by 1
18	New Zealand winter	10	1	Sunday	0	Hour advanced 1
50	New Zealand summer	3	15	Sunday	1	Hour retarded by 1
19	Palestine winter	4	15	Friday	0	Hour advanced 1
51	Palestine summer	10	15	Friday	1	Hour retarded by 1
20	Paraguay winter	9	1	Sunday	0	Hour advanced 1
52	Paraguay summer	4	1	Sunday	1	Hour retarded by 1
21	Russia winter	3	25	Sunday	2	Hour advanced 2

***DSTEvent* Value**	***Interpretation***	***Month***	***Date***	***Day of week***	***Hour***	***Action taken***
53	Russia summer	10	25	Sunday	4	Hour retarded by 2
22	Syria winter	4	1	Any	0	Hour advanced 1
54	Syria summer	10	1	Any	1	Hour retarded by 1
23	Tasmania winter	10	1	Sunday	0	Hour advanced 1
55	Tasmania summer	3	1	Sunday	1	Hour retarded by 1
24	Tonga winter	11	1	Sunday	0	Hour advanced 1
56	Tonga summer	1	25	Sunday	1	Hour retarded by 1
25	United States winter	4	1	Sunday	0	Hour advanced 1
57	United States summer	10	25	Sunday	1	Hour retarded by 1
26	Western Europe winter	3	25	Sunday	1	Hour advanced 1
58	Western Europe summer	10	25	Sunday	2	Hour retarded by 1

Bluetooth Device Classes

The 3-byte Bluetooth device class, specified using the Configure ToothPIC – Device Class command, determines what the module claims to be when other Bluetooth devices ask it. It affects the icon that appears on other Bluetooth devices and may also affect the device discovery function. In particular some mobile phones only look for certain sub classes, *e.g.* headsets.

The device class consists of three elements: the services available, the major device class and the minor device class. ToothPIC can be programmed to claim to be capable of any number of services, however exactly one Major Class must be specified. The minor device class is an optional addition, defining a subset of the major device class.

Services and Major Device Class

The first two bytes of the device class contain the services information and the major device class. They are calculated by adding together as many services that are required and the one Major Device Class required.

Byte A	***Byte B***	***Description***	***Data Type***
0x00	0x20	Limited discovery mode (default)	Services
0x01	0x00	Positioning	
0x02	0x00	Network (default)	
0x04	0x00	Rendering	
0x08	0x00	Capturing	
0x10	0x00	Object transfer (default)	
0x20	0x00	Audio	
0x40	0x00	Telephony (default)	
0x80	0x00	Information	
0x00	0x01	Computer	Device Major Class
0x00	0x02	Phone (default)	
0x00	0x03	LAN	
0x00	0x04	AV	
0x00	0x05	Peripheral	
0x00	0x06	Imaging	
0x00	0x1F	Uncategorized	
0x00	0x00	Miscellaneous Device Class	

Example

If BlueMatik is required to claim network and object transfer and information services, and appear as a peripheral then the device configuration bytes are:

Byte A	Byte B	
0x02 (0000 0010)	0x00 (0000 0000)	Network Services
0x10 (0001 0000)	0x00 (0000 0000)	Object transfer Services
0x80 (1000 0000)	0x00 (0000 0000)	Information Services
0x00 (0000 0000)	0x05 (0000 0101)	Peripheral Device Major Class

Resulting bytes (adding together the above)

Byte A	Byte B
0x92 1001 0010	0x05 0000 0101

Minor device class

The last byte defines the minor device class. Its interpretation depends on the major device class specified as follows.

Byte C	**Computer Major Class**	**Phone Major Class**	**LAN Major Class**	**AV Major Class**
0x00	Other	Other	LAN 0% utilized	Other
0x04	Desktop	Cellphone (default)		Wearable headset
0x08	Server	Cordless phone		Hands free device
0x0C	Laptop	Smartphone		
0x10	Handheld	Gateway / modem		Microphone
0x14	Palm-sized	ISDN		Loudspeaker
0x18	Wearable			Headphones
0x1C				Walkman
0x20			LAN 1-17% utilized	Car audio
0x24				Set top box
0x28				Hi-Fi
0x2C				VCR
0x30				Video camera
0x34				Camcorder
0x38				Monitor
0x3C				Monitor with audio
0x40			LAN 17-33% utilized	Conferencing device
0x48				Toy
0x60			LAN 33-50% utilized	
0x80			LAN 50-67% utilized	
0xA0			LAN 67-83% utilized	
0xC0			LAN 83-99% utilized	
0xE0			LAN 100% utilized	

Byte C	***Peripheral Device Class*** *Add together one † value and one ‡ value*	***Imaging Device Class*** *Add together as many values as apply*	***Uncategorized / Miscellaneous Device Class***
0x00	No keyboard or pointing device †		Uncategorized / Miscellaneous
0x00	Other ‡		
0x04	Joystick		
0x08	Gamepad ‡		
0x0C	Remote control ‡		
0x10	Sensing device ‡	Display	
0x14	Digitizer ‡		
0x18	Card reader‡		
0x1C			
0x20		Camera	
0x40	Keyboard but no pointing device †	Scanner	
0x80	Pointing device but no keyboard †	Printer	
0xC0	Keyboard and pointing device †		

Error Codes

The following error code responses may be generated. Errors marked * are always generate a response message in the form `$03 $02 Err`. How other errors are handles depends on the value of the Configure ToothPIC - On Internal Error setting as follows

On Internal Error setting	***Action***
01	Error number is indicated as flashes on the LEDs using the same code as the Breakpoint command. Pressing the button then resets ToothPIC.
02	ToothPIC resets immediately.
03	A response message is generated of the form `$03 $02 Err`

The error codes and their interpretations are as follows:

`Err` value			
Hex	***Decimal***	***Error Name***	***Interpretation***
$01	1	AT overrun	BlueMatik radio communications error
$02	2	Frame error	BlueMatik radio communications error
$05	3	BMT overrun	BlueMatik input buffer overrun – received character not stored in time
$03	4	No BlueMatik	BlueMatik radio failed to initialize
$04	5	Waiting prev	Still awaiting completion of previous command
$05	6	Buffer overrun	BlueMatik receive buffer overrun – no room left in receive buffer
$06	7	FxP UI	Operation not possible because user interface server is operating
$07	8	No FxP UI	Operation not possible because user interface server is not operating
$08	9	Tx timeout	BlueMatik transmit timeout
$09	10	Rx timeout	BlueMatik receive timeout
$0A	11	Memory fail	Failed to write and verify nonvolatile memory
$0B	12	Bad Argument	An argument passed to a ToothPIC function does not make sense
$0F	15	See note below	
$10	16	BMT busy	Connection attempt while module not idle
$11	17	Connected	Operation can't be performed while active connection exists
$12	18	BMT busy 2	Inquiry attempt while module not idle
$15	21	BMT confused	BlueMatik command unexpected or out of context
$17	23	Not connected	Command requires active serial connection
$18	24	No cancel	Operation cannot be cancelled
$19	25	DB Error	BlueMatik Link Database Error
$1C	28	Fragmented	Flash memory too fragmented – perform BlueMatik reset
$80	128	Sniff off	Sniff mode not enabled on BlueMatik
$81	129	Hold off	Hold mode not enabled on BlueMatik
$82	130	Low power off	No low power modes enabled on BlueMatik
$83	131	No sniff	Remote device does not support sniff mode
$84	132	No hold	Remote device does not support hold mode
$85	133	No low power	No low power modes supported on remote device
$86	134	Low power fail	Low power mode has not changed
$F0	240	Queue full	The message queue filled up and some messages were lost
$F1	241	Bad command	The command was not understood
$F2	242	Ping fail	Connection with client unexpectedly lost

Error code 15 is defined with the regular ToothPIC but not with the Stamp Edition. This error is occasionally generated if a remote device disconnected unexpectedly. The ToothPIC Stamp Edition automatically resets the Bluetooth module if this occurs and resumes normal processing. This may take a couple of seconds but otherwise will be transparent to the user.

Development Kit Inventory

The ToothPIC Stamp Edition Development Kit contains:

1. The Stamp Wireless Field Programmer application, `Stamp WFP.exe` and the library `Tokenizer.dll` which must be unzipped into the same directory.

2. The tutorial part II file `WirelessProg.bsp`, and the error corrected version `WirelessProg (error corrected).bsp`.

3. The tutorial part III file `ClockFlash.bsp`,.

4. The tutorial part IV file `IOComms.bsp`, and the error corrected version `IOComms (error corrected).bsp`.

5. The tutorial part V file `BTComms.bsp`, and the error corrected version `BTComms (error corrected).bsp`.

6. The tutorial part VI files `UserInt1.bsp` and `TSETestRes.FxP`.

7. The tutorial part VII files `TSETestRes.bsp`, `UserInt2.bsp`, `UI2_MainLoop.bsp`, `UI2_ProcCtlVal.bsp`, `UI2_ProcIOVal.bsp` and `UI2_ProcResponse.bsp`.

8. This documentation `ToothPIC Stamp Edition.pdf`.

FlexiPanel Designer and FlexiPanel Client software should be downloaded separately from *www.FlexiPanel.com.*

Revision History

Version	Date	Major revisions
3.0.00005	20-Jun-05	Initial release

Please note the following with ToothPIC product releases 3.0.00005:

- Phone clients are fully functional to FlexiPanel Protocol V2.3 only.
- Designer creation of service packs not yet implemented.

Glossary and Notation

Hex Notation

Throughout this document, numbers with an '0x' prefix should be assumed to be in hex. For example, 0xFF is completely equivalent to decimal 255.

In some partners' documentation, a '$' prefix is used in place of an '0x' prefix. '$FF' is equivalent to '0xFF' and decimal 255.

In some partners' documentation, a 'H' suffix is used in place of an '0x' prefix. 'FFH' is equivalent to '0xFF' and decimal 255.

Data Types

Data types are C standard data types; no floating point is used. C standard notation and calling conventions are assumed. Integers are explicitly defined as:

bool – unsigned char, zero for *false,* otherwise *true*

byte – 8 bit unsigned integer

int16 – 16 bit signed integer

int32 – 32 bit signed integer

signed char – 8 bit signed integer

uint16 – 16 bit unsigned integer

uint32 – 32 bit unsigned integer

unsigned char – 8 bit unsigned integer

word – 16 bit unsigned integer

Glossary

> *symbol* – Navigation drilldown to a particular item in a piece of software. A list of phrases separated by **>** symbols means: go to the program, menu or tab indicated by the first phrase, look for the second phrase and select it, look for the third phrase and select it, etc, until you find the item at the end of the list.

Big-Endian – see *Endian.*

Buffer – A linear region of memory designed for storing data entering from or departing to a communications channel.

Circular buffer – A 'first-in-first-out' buffer which wraps around. It has a start pointer indicating when the next byte is to be dispatched (i.e. read or transmitted) and an end pointer indicating the last piece of data to be dispatched. The start pointer advances when its data is dispatched; the end pointer advances when new data arrives. When either pointer reaches the end of the buffer it starts at the beginning again. If the end pointer catches up with the start pointer, the buffer is full and a *buffer* overrun occurs.

<*CR*> – The ASCII carriage return character 0x0D.

CTS – 'Clear to Send' flow control input to a DTE serial device to tell it that it is OK to transmit on its TxD line. In FlexiPanel 3.0 documentation, all devices are DTE devices and CTS on one device is connected to RTS on the corresponding device.

DTE – 'Data Terminal Emulator'. A serial device whose TxD line is a data output, *RxD* line is a data input, etc. In FlexiPanel 3.0 documentation, all devices are *treated* as DTE devices. A PC's serial port is DTE. The opposite is DCE.

DCE – 'Data Circuit Equipment'. A serial device whose TxD line is a data input, RxD line is a data output, etc. In FlexiPanel 3.0 documentation, all devices are treated as DTE devices, not DCE devices.

FlexiPanel client – Hardware or software that creates a control panel when requested to by a FlexiPanel server.

Endian – Refers to the order in which multibyte integers are stored and/or transmitted. In *Little-Endian* format, bytes are in increasing order of significance, least significant byte first. In *Big-Endian* format, bytes are in increasing order of significance, most significant byte first. In general, Flexipanel Ltd uses little-endian format, but there are exceptions.

FlexiPanel server – Hardware or software that requests a control panel to be created on a FlexiPanel client.

IC – Integrated *Circuit.*

KIPS – thousand instruction cycles per second.

<*LF*> – The ASCII line feed character 0x0A.

Little-Endian – see Endian.

LSB – least significant bit or byte, depending on context.

MIPS – million instruction cycles per second.

MSB – most significant bit or byte, depending on context.

OS – Operating System.

Overrun – A circular buffer overruns if an attempt is made to add more data to it when it is full (see *definition of circular buffer).*

PWM – Pulse Width Modulation.

RTS – 'Request to Send' flow control output from a DTE serial device to indicate that it is OK to send it data on its RxD line. In FlexiPanel 3.0 documentation, all devices are DTE devices and RTS on one device is connected to CTS on the corresponding device.

RxD – 'Receive Data' serial input to a DTE serial device. In FlexiPanel 3.0 documentation, all devices are DTE devices and RxD on one device is connected to TxD on the corresponding device.

RxD – 'Transmit Data' serial output from a DTE serial device. In *FlexiPanel* 3.0 documentation, all devices are DTE devices and TxD on one device is connected to RxD on the corresponding device.

Underrun – A circular buffer underruns if an attempt is made to dispatch data from it when it is empty.

Unicode – Two-byte integer array representing text characters. ASCII characters keep the same values in the Unicode character set.

User – The person using the finished product (as opposed to the *Developer).*

Zero Terminator – A zero-valued character used to indicate the end of a string of characters.

Legal Notices

If any of this is not clear, contact FlexiPanel Ltd for clarification.

General

FlexiPanel technology should not be used in life critical devices without the permission FlexiPanel Ltd.

FlexiPanel Ltd makes every effort to ensure, but cannot warrant, that its products and documentation are without errors and omissions. However FlexiPanel Ltd does not accept liability for consequent loss or injury as a result of using its products or interpreting its documentation. FlexiPanel Ltd will not be responsible for any third party patent infringements arising from the use of its products.

FlexiPanel Ltd reserves the right to make changes to its technology and documentation in order to improve reliability, function or design.

Software Libraries

FlexiPanel Ltd provides software such as the ToothPIC Services exclusively for use with products made by FlexiPanel Ltd. It is not permitted to use the libraries except with products made by FlexiPanel Ltd. It is not permitted to reverse engineer the security features designed to ensure that the library only works with products made by FlexiPanel Ltd.

Bluetooth Trademark

The Bluetooth trademarks are owned by Bluetooth SIG, Inc., U.S.A.

FlexiPanel Protocol

The FlexiPanel protocol and the products which use it are protected by pending patents and copyright law.

The FlexiPanel protocol allows *servers* to create user interfaces on remote *clients.*

Client software and products are freely distributable as far as we are concerned and you can do with them what you like. You can also freely produce your own client software and products which use the FlexiPanel protocol.

We make a living from licensing the FlexiPanel servers and providers of FlexiPanel server products must pay us an agreed license fee. If you buy FlexiPanel hardware products from FlexiPanel Ltd, this license is implicit. You may, under license, also make your own hardware or software FlexiPanel server products – contact us for details.

Contact Details

Sales

ToothPIC Stamp Edition is distributed and supported by:

Parallax Inc
599 Menlo Dr, Suite 100
Rocklin CA 95765, USA
sales: 888 512 1024
tel: 916 624 8333
tech support email: support@parallax.com
http://www.parallax.com

Other ToothPIC products are distributed by:

R F Solutions Ltd
Unit 21, Cliffe Industrial Estate,
Lewes, E. Sussex BN8 6JL, United Kingdom
email : sales@rfsolutions.co.uk
http://www.rfsolutions.co.uk
Tel: +44 (0)1273 898 000 Fax: +44 (0)1273 480 661

Technical Information and Customization Contact Details

ToothPIC is owned and designed by FlexiPanel Ltd. For technical support, contact FlexiPanel Ltd:

FlexiPanel

FlexiPanel Ltd
Suite 120, Westbourne Studios
242 Acklam Road
London W10 5JJ, United Kingdom
www.flexipanel.com
Tel +44 (0) 20 7524 7774
email: support@flexipanel.com