Today I had a bit of a headache and was feeling cramped in my office so I decided to escape to the treadmill for a half-hour or so in the complex's private gym. To keep it private, members gain access by waving a security card in front of a small plate adjacent to the front door – if the card is a match, cha-ching, the door unlocks and you're in.

Okay, what's going on with the card? You've probably seen them, they're everywhere. The cards in question contain technology called RFID: Radio Frequency Identification. Even if you haven't heard of RFID, you may have unknowingly been exposed to it. RFID tags can be as small and nearly as thin as a postage stamp, and are often used to track package movement in retail stores (big companies like Wal-Mart, Target, and others are adopting the technology). Drug companies are even putting RFID tags into their packaging to the prevent piracy of expensive medicines. RFID is big news, and now you can get in on it too.
There are two essential components in an RFID system: a transceiver (reader) and a transponder (tag). If it were only that simple... Tags can be active (contain their own power source) or be passive (create parasitic power from the reader's RF field). Further, tags can be read-only or read-write. Zoiks. Let's just keep things simple, shall we?

Parallax worked with world-famous hardware hacker and engineer-extraordinaire, Joe Grand (owner of Grand Idea Studio), to create a low-cost RFID reader that would be simple to use in hobbyist and professional projects. The result is a fully integrated reader PCB that contains the required circuitry and matched antenna to work with passive, read-only RFID tags. Note that the reader is specifically designed for tags that contain low-frequency (125 kHz) RFID components from EM Microelectronic. Parallax carries a couple tag types (disc and ISO card) that are manufactured by Sokymat and that meet the requirements of the reader. Figure 120.1 shows a few sample tags from Sokymat that I played with; you can clearly see the disc (far left) and ISO card (far right) tags.

![Figure 120.1: Various Sokymat RFID Tags sold by Parallax](image-url)
Is It Magic?

Okay, how does it work? It's not magic; in fact it's not terribly complicated. When power (5 VDC) is applied to the reader a green LED will indicate that it's ready to function. By pulling the ENABLE pin low, the reader becomes active (LED changes to red) and the antenna broadcasts a modulated signal. If a tag is within range (up to four inches with the Parallax reader), it will harvest the RF energy with its own antenna and modulate its unique identification code in a manner that can be detected by the reader. A microcontroller on the reader tests the bits from the RFID tag to make sure the information is valid, and then the tag number is converted to an ASCII stream to be transmitted on the SOUT pin at 2400 baud.

Keep in mind that when the antenna is active (red LED), the device is broadcasting and consuming about 200 mA from the power supply. If you're going to do a project that involves batteries, you may want to add a physical button to activate the reader only when a card is actually present, or use a timeout with SERIN (BS2 family only) to disable the reader periodically and reduce the load on the power supply.

Since RFID is so common in controlled-access and security systems, let's go that direction. And just for fun, let's build a super-simple, single-tag access control device with a BS1. Can we do it? Absolutely. In fact, the code is so simple we can look at the whole thing in one shot:

```
Main:
  LOW Enable
  SERIN RX, T2400, ($0A, "0F0184F20B")
  HIGH Enable

Access Granted:
  HIGH Latch
  PAUSE 2000
  LOW Latch
  GOTO Main
```

We start by activating the reader (ENABLE pin is pulled low) and then simply waiting for a specific tag string. And let me correct something I left out: the tag ID string is preceded by a linefeed character ($0A) and followed by a carriage return ($0D). We'll see why this is useful a bit later.
Figure 120.2: RFID to BASIC Stamp Connection
In this program, SERIN does all the work. We construct the SERIN line to wait for the linefeed character, then the specific characters in the valid RFID tag string. Once that shows up the program drops to the point called Access_Granted where we activate an output that will do what we need it to do. We could, for example, disable an electric door lock that gives us – and just us – access to something special. After a brief delay the lock-control output is enabled and we go back to the top.

The logical question is, "Where did you get the tag ID string?" From the tag, of course. I just mentioned that the (ASCII) tag string is preceded by a linefeed and followed by a carriage return. We can put this to use by connecting the reader to a terminal program. Note that we need to go through an RS-232 line driver (e.g., MAX232, DS275, etc.) as the serial output is at TTL levels. Figure 120.3 shows the connections and Figure 120.4 shows the output when using a manually-opened terminal from the BASIC Stamp IDE (note that the baud rate is set to 2400). In most cases we'll use a BASIC Stamp to work with the reader, but be aware that you can also connect directly to a custom PC application using a simple interface as shown.

![Figure 120.3: RFID to Serial Port Connectivity](image-url)
By the way, if you happen to have the new Parallax Serial LCD module, you can use it as a terminal – and you don't need a level-shifter. Simply set the LCD Mode switches for 2400 baud (1 = Up, 2 = Down) and connect the RFID reader's SOUT pin to the LCD's RX pin. And don't worry if it's not convenient for you to connect the RFID reader to a terminal or LCD – we can always use a BASIC Stamp module to read and display an unknown tag string.
Open Sesame

Most security systems will have more than one legal user, so let's update the program to work with multiple tags. To be honest with you, I had to go back to the BS1 manual on several occasions for this program because the BS1 – while very cool – is not quite as convenient as its big brother the BS2. It's very inexpensive though (especially with OEM parts), and is worth considering for a low-cost access control system.

After we've recorded our tag IDs we can put them into a simple EEPROM-based table for storage. Here's what my table looks like:

Tags:
- EEPROM ("0F0184F20B")
- EEPROM ("0F01D9D263")
- EEPROM ("04129C1B43")
- EEPROM ("0000000000")
- EEPROM ("0000000000")

Remember that your table will be different as the RFID tag strings are unique. By using the Memory Map feature of the BASIC Stamp Editor IDE I found I had room for five tags. Since I only had three to work with I padded the table for the unused positions. A program constant will prevent us from searching past the known tag strings.

The next step is to setup our control outputs. For this security program we will turn off the reader and lock the door.

Reset:
- HIGH Enable
- LOW Latch

After basic setup we drop into the heart of the program where the reader is activated and we wait on a tag ID string. Now you'll see what I was just talking about with the BS1 not being quite as convenient as the BS2 – there is no STR modifier with the BS1 and its memory cannot be treated like an array; everything must be done one byte at a time.

Main:
- LOW Enable
- SERIN RX, T2400, ($0A),
  - tag0, tag1, tag2, tag3, tag4,
  - tag5, tag6, tag7, tag8, tag9
- HIGH Enable
Note that the SERIN line is really long, but needs to be on one line to make sure that all 11 bytes (header plus 10) received are in fact received properly. For publication, I've split the line, but you'll find it all together in the downloadable version of the code.

With the tag ID stored in the BS1's RAM, we can compare it to our table entries to determine whether the tag presented is a match or not. Admittedly, this looks a little hairy, but it's really not that bad. To keep things concise, I'm only showing the first and last bytes, but the same code is required for all ten elements of the RFID tag string.

```
Check_List:
  FOR tagNum = 0 TO LastTag
    pntr = tagNum * 10 + 0
    READ pntr, char
    IF char <> tag0 THEN Bad_Char

    ' removed for clarity
    pntr = tagNum * 10 + 9
    READ pntr, char
    IF char <> tag9 THEN Bad_Char
    GOTO Tag_Found
  GOTO Next

Bad_Char:
  NEXT

Tag_Found:
  DEBUG "Entry: ", #tagNum, CR
  HIGH Latch
  SOUND Spkr, (114, 165)
  LOW Latch
  GOTO Main
```

As you can see, comparing each byte of the RFID string against a table entry requires three steps that are placed in a loop: 1) We create a pointer to the corresponding position of the table entry, 2) We read the character from the table, and then 3) We compare the two bytes. If they don't match the program jumps to the label called Bad_Char and we'll either move to the next table entry, or if at the end of the table we'll fall through the loop.

Let's say we do have a match. When that occurs we will jump to the label called Tag_Found and execute the door-opening code:

```
Tag_Found:
  DEBUG "Entry: ", #tagNum, CR
  HIGH Latch
  SOUND Spkr, (114, 165)
  LOW Latch
  GOTO Main
```
The latch is activated (to allow entry) and then a beep is played through piezo speaker of amplifier circuit to alert the user. The beep stops after about two seconds and the door relocks. With that we go back to the top of the program.

When the tag presented does not match any of our table entries a one-second groan is played through the speaker:

Bad_Tag:
  DEBUG "Unknown Tag", CR
  SOUND Spkr, (25, 80)
  PAUSE 1000
  GOTO Main

And that's that. As I told you, it's really very simple – the trickiest part about this program is working around the behaviors of the BS1; but even that wasn't so bad. Okay, let's port this baby to the BS2.

Starting back at the tags table, we're going to add names to each tag. In our demo program we'll just send these to the Debug Terminal window, but we could just as easily put them on an LCD if we ever decide to add one to the project.

<table>
<thead>
<tr>
<th>Tag</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag1</td>
<td>&quot;0F0184F20B&quot;</td>
</tr>
<tr>
<td>Tag2</td>
<td>&quot;0F01D9D263&quot;</td>
</tr>
<tr>
<td>Tag3</td>
<td>&quot;04129C1B43&quot;</td>
</tr>
<tr>
<td>Name0</td>
<td>&quot;Unauthorized&quot;, CR, 0</td>
</tr>
<tr>
<td>Name1</td>
<td>&quot;George W. Bush&quot;, CR, 0</td>
</tr>
<tr>
<td>Name2</td>
<td>&quot;Dick Cheney&quot;, CR, 0</td>
</tr>
<tr>
<td>Name3</td>
<td>&quot;Condoleezza Rice&quot;, CR, 0</td>
</tr>
</tbody>
</table>

Note that the name strings are zero-terminated so that we're not restricted to a specific length. We'll get to the printing routine later.

Did you read last month's column? If not, why not?! Okay, I'll put my bruised ego aside and just point out that we're going to take advantage of the lessons on conditional compilation in this program. When I stated above that we would port the program to the BS2, I meant the BS2 family; the entire BS2 family.

The first thing to consider is the RAM required to read the RFID tag string: 10 bytes. This is usually temporary in nature and it would really be nice if we didn't have to use our variable space to handle it. Well, if we use the BS2p or BS2pe we don't have to; we can use the
Scratchpad as a serial buffer. The first thing we have to do, though, is setup the program so that the compiler can detect a BS2p or BS2pe and create a symbol to that effect.

`#DEFINE __No_SPRAM = ($STAMP < BS2P)`

With this definition the symbol called __No_SPRAM will be set to True if we're not using a BS2p or BS2pe – hence are forced to define a buffer in our variable RAM space. Let's get to that:

```c
#IF __No_SPRAM #THEN
    buf VAR Byte(10)
#ELSE
    chkChar VAR Byte
#ENDIF
```

Remember that conditional compilation is just that: conditional, and it means that what actually gets compiled and downloaded will change based on the BASIC Stamp module we're using. In the code above, the array called buf is only created when using a BS2, BS2e, or BS2sx. When using a BS2p or BS2pe, we create a variable called chkChar.

And now to the core of the program. What you'll realize is that no matter which BS2-family module we use, receiving the RFID tag string in this program is much easier here than with the BS1. The only question is where those bytes will be stored, and that is determined by the module in use.

```c
Main:
    LOW Enable
    #IF __No_SPRAM #THEN
        SERIN RX, T2400, [WAIT($0A), STR buf\10
    #ELSE
        SERIN RX, T2400, [WAIT($0A), SPSTR 10]
    #ENDIF
    HIGH Enable
```

When we're using a BS2, BS2e, or BS2sx the tag string is stored in our variable buffer, otherwise it gets stuffed into the first 10 bytes of the Scratchpad. But see how much easier this is? The STR and SPSTR modifiers are huge timesavers here.

With the tag string in RAM, we can compare it against the table, and again, it's much easier with the BS2-family.
Check_List:
FOR tagNum = 1 TO LastTag
  FOR idx = 0 TO 9
    READ (tagNum - 1 * 10 + idx), char
    #IF __No_SPRAM #THEN
      IF (char <> buf(idx)) THEN Bad_Char
    #ELSE
      GET idx, chkChar
      IF (char <> chkChar) THEN Bad_Char
    #ENDIF
  NEXT
GOTO Tag_Found
Bad_Char:
  NEXT

As with the BS1, a loop is used to work through the known tag strings. What we're able to do here, however, is use a second loop to test each byte of the received string. The inner loop reads the appropriate byte from the current tag data and compares it against the corresponding tag byte from the reader. Our conditional symbol sets up the code to make the comparison against a byte in the variable array or against a byte from the Scratchpad. Note that the Scratchpad cannot be treated like an array so we are forced to use GET to access the appropriate byte.

Okay, moving on to a good tag we will do the same as before: sound the beeper (with FREQOUT) and disable the security lock. Remember that FREQOUT is one of those instructions that differs from one BASIC Stamp model to another, so conditional compilation constants are used to keep the timing (two seconds) and tone (880 Hz) the same, no matter which module we use.

Tag_Found:
  GOSUB Show_Name
  HIGH Latch
  FREQOUT Spkr, 2000 */ TmAdj, 880 */ FrAdj
  LOW Latch
  GOTO Main

We've also added the ability to display the name of the person who is assigned to the valid tag. A simple loop will send the characters in the name to a display – we'll keep it easy and use the Debug Terminal.
Show_Name:
    DEBUG DEC tagNum, ": ":
    LOOKUP tagNum,
        [Name0, Name1, Name2, Name3], idx
    DO
        READ idx, char
        IF (char = 0) THEN EXIT
        DEBUG char
        idx = idx + 1
    LOOP
    RETURN

The result of the tag search (in the variable tagNum) will be from one to the number of known tags if the tag string is valid – if not, the search will result in zero. The Show_Name routine uses the result of the tag search to LOOKUP the first character of the corresponding name. After that, each character is printed in a loop until the zero terminator is encountered. We could very easily change the DEBUG line to SEROUT for a serial LCD, or to LCDOUT if we're using a BS2p or BS2pe and have a parallel LCD connected as required.

Well, that's it. That was pretty simple, wasn't it? I think so, and I'm having a lot of fun with the RFID reader. Be sure to check out the web resources listed, there's lots of interesting information on RFID technology, and Joe Grand's web site has some really cool stuff (if you're into hardware hacking, you'll love his books).

Until next time ... Happy Stamping!

Web Resources

www.parallax.com
www.grandideastudio.com
www.rfidjournal.com
www.emmicroelectronic.com
www.sokymat.com
File....... RFID_Reader.BS1
Purpose.... RFID Tag Reader
Author..... Jon Williams, Parallax
E-mail..... jwilliams@parallax.com
Started....
Updated.... 07 FEB 2005

{$STAMP BS1}
{$PBASIC 1.0}

=====================================================================

' -----[ Program Description ]---------------------------------------------
' Reads and displays RFID tag strings.

' -----[ Revision History ]-------------------------------------------------

' -----[ I/O Definitions ]--------------------------------------------------
SYMBOL  Enable          = 0                     ' low = reader on
SYMBOL  RX              = 1                     ' serial from reader

' -----[ Constants ]-------------------------------------------------------

' -----[ Variables ]-------------------------------------------------------

' -----[ EEPROM Data ]-----------------------------------------------------

' -----[ Initialization ]---------------------------------------------------

' -----[ Program Code ]-----------------------------------------------------
Main:
  LOW Enable
    ' wait for header ($0A), then accept 10 RFID bytes
  SERIN RX, T2400, ($0A),B0,B1,B2,B3,B4,B5,B6,B7,B8,B9
  HIGH Enable
Show_Tag:
DEBUG "Tag ID: ", @@B0,@@B1,@@B2,@@B3,@@B4,@@B5,@@B6,@@B7,@@B8,@@B9
DEBUG CR
PAUSE 1000                                         ' time to remove tag
GOTO Main

' -----[ Subroutines ]-----------------------------------------
File....... RFID_Single.BS1
Purpose.... Single-tag RFID Tag Reader
Author..... Jon Williams, Parallax
E-mail..... jwilliams@parallax.com
Started....
Updated.... 07 FEB 2005

{%STAMP BS1%
{SPBASIC 1.0%

=====================================================================

-----[ Program Description ]---------------------------------------------

Looks for a specific RFID tag and when found, opens an electric lock

-----[ Revision History ]------------------------------------------------

-----[ I/O Definitions ]-------------------------------------------------

SYMBOL Enable = 0                 ' low = reader on
SYMBOL RX    = 1                 ' serial from reader
SYMBOL Latch  = 2                 ' lock/latch control

-----[ Constants ]------------------------------------------------------

-----[ Variables ]------------------------------------------------------

-----[ EEPROM Data ]----------------------------------------------------

-----[ Initialization ]--------------------------------------------------

Reset:
HIGH Enable                 ' turn of RFID reader
LOW Latch                   ' lock the door!

-----[ Program Code ]----------------------------------------------------

Main:
LOW Enable                 ' activate the reader
SERIN RX, T2400, ($0A, "0F0184F20B") ' wait for header & tag

The Nuts and Volts of BASIC Stamps (Volume 6) • Page 75
<table>
<thead>
<tr>
<th>COMMAND</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH Enable</td>
<td>' deactivate reader</td>
</tr>
<tr>
<td>Access Granted:</td>
<td></td>
</tr>
<tr>
<td>HIGH Latch</td>
<td>' open the lock</td>
</tr>
<tr>
<td>PAUSE 2000</td>
<td>' -- for two seconds</td>
</tr>
<tr>
<td>LOW Latch</td>
<td>' relock</td>
</tr>
<tr>
<td>GOTO Main</td>
<td>------------------------------</td>
</tr>
</tbody>
</table>

' -----[ Subroutines ]-----------------------------------------------
File....... RFID.BS1
Purpose.... RFID Tag Reader / Simple Security System
Author..... Jon Williams, Parallax
E-mail..... jwilliams@parallax.com
Started....
Updated.... 07 FEB 2005

{$STAMP BS1}
{$PBASIC 1.0}

=====================================================================

-----[ Program Description ]---------------------------------------------

Reads tags from a Parallax RFID reader and compares to known tags (stored
in EEPROM table). If tag is found, the program will disable a lock.

-----[ Revision History ]-----------------------------------------------

-----[ I/O Definitions ]-----------------------------------------------
SYMBOL  Enable          = 0                     ' low = reader on
SYMBOL  RX              = 1                     ' serial from reader
SYMBOL  Latch           = 2                     ' lock/latch control
SYMBOL  Spkr            = 3                     ' speaker output

-----[ Constants ]-------------------------------------------------------
SYMBOL  LastTag         = 2                     ' 3 tags; 0 to 2

-----[ Variables ]-------------------------------------------------------
SYMBOL  tag0            = B0                    ' RFID bytes buffer
SYMBOL  tag1            = B1
SYMBOL  tag2            = B2
SYMBOL  tag3            = B3
SYMBOL  tag4            = B4
SYMBOL  tag5            = B5
SYMBOL  tag6            = B6
SYMBOL  tag7            = B7
SYMBOL  tag8            = B8
SYMBOL  tag9            = B9
SYMBOL  tagNum          = B10                   ' from EEPROM table
Symbolic constants and tags:

Symbolic constants:

```
SYMBOL pntr = B11            ' pointer to char in table
SYMBOL char = B12            ' character from table
```

Tags:

```
EEPROM ("0F0184F20B")         ' valid tags
EEPROM ("0F01D9D263")
EEPROM ("04129C1B43")
EEPROM ("0000000000")         ' space for other tags
EEPROM ("0000000000")
EEPROM ("0000000000")
```

Initialization:

```
Reset:
    HIGH Enable               ' turn off RFID reader
    LOW Latch                 ' lock the door!

Main:
    LOW Enable
    ' wait for header, then accept 10 RFID bytes
    SERIN RX, T2400, ($0A),tag0,tag1,tag2,tag3,tag4,tag5,tag6,tag7,tag8,tag9
    HIGH Enable

    Check_List:
        FOR tagNum = 0 TO LastTag  ' scan through known tags
            pntr = tagNum * 10 + 0 : READ pntr, char  ' read char from DB
            IF char <> tag0 THEN Bad_Char
            pntr = tagNum * 10 + 1 : READ pntr, char  ' compare with tag data
            IF char <> tag1 THEN Bad_Char
            pntr = tagNum * 10 + 2 : READ pntr, char
            IF char <> tag2 THEN Bad_Char
            pntr = tagNum * 10 + 3 : READ pntr, char
            IF char <> tag3 THEN Bad_Char
            pntr = tagNum * 10 + 4 : READ pntr, char
            IF char <> tag4 THEN Bad_Char
            pntr = tagNum * 10 + 5 : READ pntr, char
            IF char <> tag5 THEN Bad_Char
            pntr = tagNum * 10 + 6 : READ pntr, char
            IF char <> tag6 THEN Bad_Char
            pntr = tagNum * 10 + 7 : READ pntr, char
            IF char <> tag7 THEN Bad_Char
            pntr = tagNum * 10 + 8 : READ pntr, char
```
IF char <> tag8 THEN Bad_Char
pntr = tagNum * 10 + 9 : READ pntr, char
IF char <> tag9 THEN Bad_Char
    GOTO Tag_Found                          ' all match -- good tag
Bad_Char: NEXT

Bad_Tag:
DEBUG "Unknown Tag", CR
SOUND Spkr, (25, 80)                             ' groan
PAUSE 1000
GOTO Main

Tag_Found:
DEBUG "Entry ", #tagNum, CR
HIGH Latch                                     ' remove latch
SOUND Spkr, (114, 165)                          ' beep
LOW Latch                                      ' restore latch
GOTO Main

END

' -----[ Subroutines ]------------------------------------------
' File....... RFID.BS2
' Purpose.... RFID Tag Reader / Simple Security System
' Author..... Jon Williams, Parallax
' E-mail..... jwilliams@parallax.com
' Started....
' Updated.... 07 FEB 2005
'
' {$STAMP BS2e}
' {$PBASIC 2.5}
'==========================================================================

' -----[ Program Description ]---------------------------------------------
' Reads and displays RFID tag strings.
'--------------------------------------------------------------------------

' -----[ Revision History ]--------------------------------------------------

' -----[ I/O Definitions ]---------------------------------------------------
Enable          PIN     0                       ' low = reader on
RX              PIN     1                       ' serial from reader

' -----[ Constants ]--------------------------------------------------------
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T2400       CON     396
  #CASE BS2SX, BS2P
    T2400       CON     1021
#ENDSELECT

' -----[ Variables ]--------------------------------------------------------
buf             VAR     Byte(10)                ' RFID bytes buffer

' -----[ EEPROM Data ]------------------------------------------------------

' -----[ Initialization ]----------------------------------------------------

' -----[ Program Code ]------------------------------------------------------
Main:
  LOW Enable                                    ' activate the reader
  SERIN RX, T2400, [WAIT(§0A), STR buf\10]      ' wait for hdr + ID
  HIGH Enable                                    ' deactivate reader
Show_Tag:
  DEBUG "Tag ID: ", STR buf\10, CR             ' display ID
  PAUSE 1000                                    ' time to remove tag
  GOTO Main

' -----[ Subroutines ]-------------------------------
' File....... RFID.BS2
' Purpose.... RFID Tag Reader / Simple Security System
' Author..... Jon Williams, Parallax
' E-mail..... jwilliams@parallax.com
' Started....
' Updated.... 07 FEB 2005
'{
' {$STAMP BS2}
' {$PBASIC 2.5}
'
' =========================================================================

-----[ Program Description ]---------------------------------------------
'R Reads tags from a Parallax RFID reader and compares to known tags (stored
'in EEPROM table). If tag is found, the program will disable a lock.

-----[ Revision History ]-------------------------------------------------

-----[ I/O Definitions ]---------------------------------------------------
Enable   PIN    0             ' low = reader on
RX       PIN    1             ' serial from reader
Latch    PIN    2             ' lock/latch control
Spkr     PIN    3             ' speaker output

-----[ Constants ]--------------------------------------------------------
#SELECT $STAMP
#CASE BS2, BS2E
   T2400   CON    396            ' reader baud rate
   TmAdj   CON    $100           ' x 1.0 (time adjust)
   FrAdj   CON    $100           ' x 1.0 (freq adjust)
#CASE BS2SX
   T2400   CON    1021
   TmAdj   CON    $280           ' x 2.5
   FrAdj   CON    $066           ' x 0.4
#CASE BS2P
   T2400   CON    1021
   TmAdj   CON    $3C5           ' x 3.77
   FrAdj   CON    $044           ' x 0.265
#CASE BS2PE
   T2400   CON    396
   TmAdj   CON    $100           ' x 1.0
   FrAdj   CON    $0AA           ' x 0.665
LastTag CON 3 ' three known tags

#define __no_spram = ($stamp < bs2p) ' does module have SPRAM?

' -----[ Variables ]---------------------------------------------------------------
#if __no_spram #then
buf VAR Byte(10) ' RFID bytes buffer
#else
chkChar VAR Byte ' character to test
#endif

tagNum VAR Nib ' from EEPROM table
idx VAR Byte ' tag byte index
char VAR Byte ' character from table

' ----- [ EEPROM Data ]-------------------------------------------------------------
tag1 DATA "0F0184F20B" ' valid tags
tag2 DATA "0F01D9D263"
tag3 DATA "04129C1B43"

name0 DATA "Unauthorized", CR, 0
name1 DATA "George W. Bush", CR, 0
name2 DATA "Dick Cheney", CR, 0
name3 DATA "Condoleeza Rice", CR, 0

' ----- [ Initialization ]------------------------------------------------------------
reset:
HIGH Enable ' turn of RFID reader
LOW Latch ' lock the door!

' ----- [ Program Code ]-------------------------------------------------------------
main:
LOW Enable ' activate the reader
#if __no_spram #then
serin rx, t2400, [wait($0a), str buf\10] ' wait for hdr + ID
#else
serin rx, t2400, [wait($0a), spstr 10]
#endif
HIGH Enable ' deactivate reader
Check_List:
    FOR tagNum = 1 TO LastTag                     ' scan through known tags
        FOR idx = 0 TO 9                            ' scan bytes in tag
            READ (tagNum - 1 * 10 + idx), char        ' get tag data from table
            #IF __No_SPRAM #THEN
                IF (char <> buf(idx)) THEN Bad_Char  ' compare tag to table
            #ELSE
                GET idx, chkChar                        ' read char from SPRAM
                IF (char <> chkChar) THEN Bad_Char      ' compare to table
            #ENDIF
            NEXT
        GOTO Tag_Found                              ' all bytes match!
    Bad_Char:                                       ' try next tag
        NEXT
    Bad_Tag:
        tagNum = 0
        GOSUB Show_Name  ' print message
        FREQOUT Spkr, 1000 */ TmAdj, 115 */ FrAdj     ' groan
        PAUSE 1000
        GOTO Main
    Tag_Found:
        GOSUB Show_Name  ' print name
        HIGH Latch       ' remove latch
        FREQOUT Spkr, 2000 */ TmAdj, 880 */ FrAdj     ' beep
        LOW Latch        ' restore latch
        GOTO Main
    END

' -----[ Subroutines ]-----------------------------------------------------------------------

' Prints name associated with RFID tag
Show_Name:
    DEBUG DEC tagNum, ": "
    LOOKUP tagNum,
        [Name0, Name1, Name2, Name3], idx      ' point to first character
        DO
            READ idx, char                          ' read character from name
            IF (char = 0) THEN EXIT                 ' if 0, we're done
            DEBUG char                             ' otherwise print it
            idx = idx + 1                          ' point to next character
        LOOP
    RETURN