

## How To Use Protocol Builder

If you have followed the instructions in the "JP1 – How Easy Is It?" document and have learned the buttons from your original remote, but when you look at them in IR.exe the signals are not decoded (i.e., the protocol and device code info is blank, or just displays "GAP" information), then you probably need to create a new protocol for this device.

Actually, your next move should really be to start a new thread on this in the JP1 Forums at <http://www.hifi-remote.com/forums> just in case someone has already developed something you can use, but assuming that that is not the case, you can probably use the Protocol Builder spreadsheet (aka "PB") to build a new protocol for your device. I say "probably" because PB is only really designed to handle the most common protocols, so if the signals for your device don't quite conform to the simple mould, they may require some custom assembler code, which is outside the scope of what PB can do without the help of a JP1 expert.

At any rate, let's proceed.

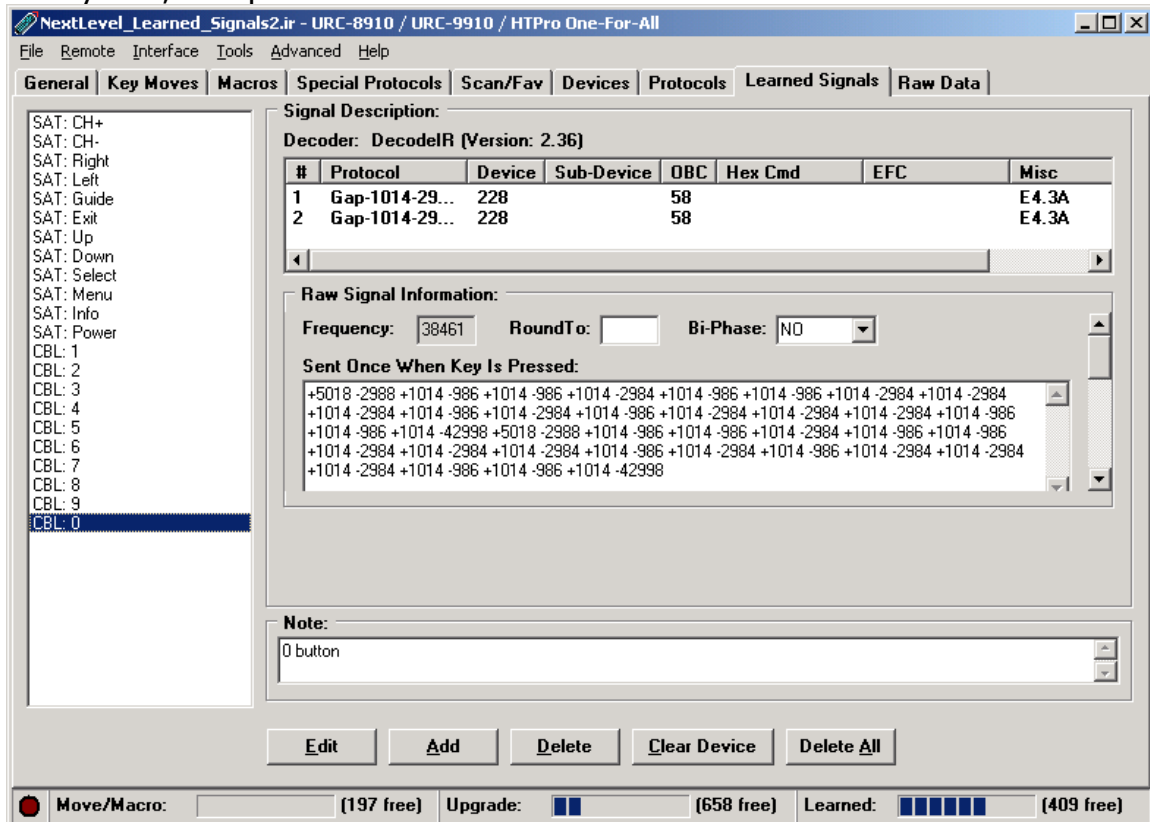


Figure 1 — the "Learned Signals" tab in IR

First, look at the signals in the "Learned Signals" tab of IR.exe. If you don't see the "Raw Signal Information" part of the "Learned Signals" tab as shown above, select "Force Learned Timings" on the Advanced Menu. Copy & paste the timing data over to a text editor (such as Notepad, etc). It would be helpful if the text editor allows you to perform global edits. The version of Notepad that ships with Windows 2000, XP or Vista does allow global edits, but earlier versions do not, in which case it might be a better idea to use Wordpad. Copy the data in the order that the buttons appear in the Learned Signals tab so you don't lose track of which button each set of data belongs to.

When you have completed copying the data, you should have a file somewhat like the codes1.txt file included in this zip file. The next step is to try and clean up the signal times. IR tends to report the signal times very exactly, but in reality most devices have quite a large level of tolerance for signals not being exactly what they should be. So, what we need to do now is try and round all the numbers up or down so all similar numbers are actually the same. For example, if you see times like +988, +995, +1015, etc, they are all really the same time and can be rounded to +1000.

Newer versions of IR.exe have the ability to round the timing data for you. In the "Raw Signal Information" area, the box labeled "RoundTo:" allows you to type in a number, and IR.exe will round all the timing numbers accordingly. For example, if you type 10 in the box, all the timing will be rounded off to numbers evenly divisible by 10.

Regardless of which method you use to round off the numbers, when you have finished, you should end up with a file somewhat like codes2.txt in the zip. The data in this file should be viewed as pairs, where a pair is a grouping of a ON time and an OFF time. The ON times are represented by positive numbers and the OFF times are represented by negative numbers. The next step is to attempt to identify the following pairs:

- Lead-in pair
- Logical "zero" pair
- Logical "one" pair
- Lead-out pair

Looking at the data in codes2.txt, you will notice that the first pair is "+5000 -3000" and then there is a mixture of the following two pairs "+1000 -1000" and "+1000 -3000", then eventually you will find a pair with a much bigger OFF time, and then the whole signal repeats. In all of these examples, the signal is repeated 2 times, and each repetition includes the lead-in and lead-out pairs.

Therefore, here's the data for the afore mentioned pairs:

- Lead-in pair = "+5000 -3000"
- Logical "zero" pair = "+1000 -1000"
- Logical "one" pair = "+1000 -3000"
- Lead-out pair = (to be determined)

Actually, there is no exact science to deciding which of the two variable pairs is the logical "one" and "zero" pairs. Generally speaking, the "zero" pair will have smaller times than the "one" pair, but even that's just a guess. Bottom line, it doesn't really matter. In fact, UEI gets it wrong more often than they get it right in their protocols.

So, now that we've guessed the "one" and "zero" pairs, let's edit the file so we can see the binary. My preferred way to do this is to edit the strings like this:

" +1000 -1000" → 0^

" +1000 -3000" → 1^

I put the ^ character after the number so that if I later decide I want to reverse the pairs and make the first pair the logical "one" I can edit all the 0^ and 1^ strings and reverse them. When you're done editing the "one" and "zero" pairs, you should end up with a file somewhat like codes3.txt You'll notice that I've inserted a space in the middle of the binary string, breaking it into two groups of 8. In the binary world, 8 is considered a round number, so you'll often find that the total length of the binary signal is a multiple of 8. In this case you should notice that the second string is constant for all the signals, so we can consider this to be our "device code". The first string is different for all the signals, so we can consider this to be our "command code".

While units of 8 are the most common, there are always exceptions to every rule. Your immediate goal is to determine which bits are constant for every signal and which bits vary. The more learned signals you have, the greater the chance that you will get this right.

At this point you should examine the data and see if you can spot any "compliments". A compliment code is where the data is the same as the previous data, only inverted. (Inverted, or complimented, means that if the original bit was 0, the new bit should be 1, and vice versa).

Normally, you would probably find that the lead-out pair is constant for all signals, but in this case it varies, which leads me to suspect that it's calculated as a result of the total signal length. Every "zero" pair lasts 2000 uSecs and every "one" pair lasts 4000 uSecs, so given that there are 16 bits in each of these signals, the data portion of the signal will last a minimum of 32000 uSecs (which would mean that the binary was all zeroes). Each "one" in the signal would add

an additional 2000 uSecs to the total. If you calculate the total duration of the data portion of the signal and add in the lead-out times, you'll find that each signal lasts for a total of 98000 uSecs.

Now, let's move on to the PB spreadsheet itself; when you first open it, it looks like this:

**Figure 2 — Protocol Builder**

And you thought KM was scary! Don't worry, just like KM, this spreadsheet is actually easier to use than it looks. Now we have to start entering the data that we have just worked on.

- 1) **Description:** Very short description of the signal; our signals were for a "Next Level" cable box, so let's enter "Next Level".
- 2) **Remote Type:** Generally speaking, you should leave this as "S3C8+ (New)" as this covers most of the remotes in use in the JP1 world. This is the correct setting for JP1.3 remotes. If you are using a JP1.2 remote, select "HCS08", or if you are using a JP1.1 remote select "SST". If you believe that you need one of the other options, but are not sure which one to use, you should post a question about it in the forums.
- 3) **Protocol ID:** Basically, this can be whatever you want, but it's generally a good idea to use an ID that is not already established as an official protocol ID. You enter the code in hex format where 0 is the smallest value allowed

and 1FF is the largest. So, I would advise that you use 1FF for this new protocol.

- 4) **Frequency:** Enter the value displayed in IR's "Learned Signals" tab. IR displays the frequency in Hertz, while PB accepts this entry in kHz (kilohertz), so you will need to divide by 1000 and round off to the nearest tenth. In our example, IR shows a frequency of 38461, so we divide by 1000 to get 38.461, and then round off to 38.5. (IR may show a slightly different frequency for each learned signal, so you should pick a value that is approximately correct for all of them.)
- 5) **Duty Cycle:** There's no easy way to determine this, so use 30% unless you have reason to do otherwise.
- 6) **Signal Structure:** This is determined by how the binary data is laid out. In our example, the command code came before the device code and there weren't any compliments involved, so we should select "cmd-dev".
- 7) **[Device] Bytes:** this is the number of bytes of fixed data. In our example we found one fixed byte, so we will enter 1.
- 8) **Bits/Dev:** this is how many bits are in each fixed byte. The max value is 8, which is also the most common answer. In our example, there are 8 fixed bits.
- 9) **[Command] Bytes:** this is the number of bytes of variable data. In our example we found one variable byte, so we will enter 1.
- 10) **Bits/Cmd:** this is how many bits are in each variable byte. The max value is 8, which is also the most common answer. In our example, there are 8 fixed bits.
- 11) **[Repeat] Value:** this is the minimum number of times the signal should repeat. It appears from the data in our examples that the signal should repeat at least 2 times.
- 12) **Type:** when set to 'Minimum', item 11 above controls the minimum number of repeats; supposedly 'Forced' makes the signal repeat that many times only, but I understand this setting is not always reliable.
- 13) **Hold:** this determines if the signal repeats, and if so, under what circumstances. "Yes" means the whole signal will repeat for as long as the button is held down. "No" means the signal does not repeat at all. Selecting "Ch+/-, Vol+/-, FF, Rew" will cause the signal to only repeat when those buttons are used, otherwise it will not repeat. The final selection is "No data bits in repeat", which is outside of the scope of this document. All of the data in our limited sample would suggest that our signal does not need to repeat when the button is held down, so "no" would be the logical selection. However, the CH+ and CH- buttons did repeat, so "Ch+/-, Vol+/-, FF, Rew" is probably the right selection. If in doubt, try each of them and see which works.

- 14) ['1' Burst] ON (µSec):** this is the ON time for the logical "one" pair in microseconds (which is the positive number), so for our example we will enter 1000.
- 15) OFF (µSec):** this is the OFF time for the logical "one" pair in microseconds (which is the negative number), so for our example we will enter 3000.
- 16) ['0' Burst] ON (µSec):** this is the ON time for the logical "zero" pair in microseconds (which is the positive number), so for our example we will enter 1000.
- 17) OFF (µSec):** this is the OFF time for the logical "zero" pair in microseconds (which is the negative number), so for our example we will enter 1000.
- 18) [Lead-In Style]:** this determines whether there is a lead-in burst pair for this signal, and if so, whether it appears once only or in every repetition of the signal. In our example, the lead-in is present in every repetition, so "Same Every Frame" is the correct selection.
- 19) [Lead-In] ON (µSec):** this is the ON time for the lead-in burst pair in microseconds, so for our example we will enter 5000.
- 20) OFF (µSec):** this is the OFF time for the lead-in burst pair in microseconds, so for our example we will enter 3000.
- 21) [Lead-Out Style]:** this determines whether the lead-out portion is a burst pair or just a single OFF time. "0=[-LO]" means it's just a single OFF time. "1=[LI,-LO]" means it's a pair which uses the Lead-in pairs ON time. "2=[OneOn, -LO]" means it's a pair that uses the logical "one" pairs ON time. "3=[LI, [OneOn, -LO]" means it's a pair where the ON time is the sum of the lead-in pairs ON time and the logical "one" pairs ON time. In our example, the lead-out ON time was 1000, which matches the ON time for the logical "one" pair, so we should select "2=[OneOn, -LO]".
- 22) [Lead-Out] OFF (µSec):** this is the lead-out OFF time in microseconds. (See below.)
- 23) OFF as Total:** when "No" is selected, the OFF time entered above will be used for every signal. When "Yes" is selected, the total duration of the data portion of the signal is calculated and subtracted from the time entered above to determine the actual lead-out time. In our example, we found that the lead-out does indeed vary based on the total length of the signal and we also found that the total length of the signal is always 98000 microseconds, so we should select "Yes" here and enter 98000 as the Lead-Out OFF time.

You will notice that I skipped over some of the selections available in PB; any skipped item should be left set to its default value. Use of these skipped items is outside of the scope of this document, where I'm trying to keep things *somewhat* simple.

With all the above selections made, your PB screen should look like this...

The screenshot shows the Protocol Builder (PB) software interface. The main window is titled "Microsoft Excel - protocol-builder-v401.xls". The interface is divided into several sections:

- Top Bar:** Contains buttons for "NEW File", "LOAD File...", "SAVE File...", and "UNDO".
- Left Panel:** A list of protocols with "Next Level" selected. Below the list are various configuration options for the selected protocol, including:
  - Remote Type: 53C8+ (New)
  - Protocol ID: 1FF
  - Frequency (Khz): 38.500
  - Duty Cycle %: 30
  - Signal Structure: cmd-dev
  - [Device] Bytes: 1
  - Bits/Dev: 8
  - Dev Bit Doubling: None
  - [Command] Bytes: 1
  - Bits/Cmd: 8
  - Cmd Bit Doubling: None
  - [Repeat] Value: 2
  - Type: Minimum
  - Hold: Ch+/-, Vol+/-, FF, Rew
  - [Check Byte] Style: None
  - # Bytes Checked: 1
  - Mini-Combiner: None
  - Signal Style: LSB-Comp
- Right Panel:** Contains a large text area for the protocol code. The code is:
 

```
Upgrade protocol 0 = 01 FF (53C8+) Next Level (PB
3C 90 11 8B 14 F5 56 08 08 01 F4 05 C8 01 F4 01
E0 BF 68 09 C4 05 C8 FF 02 8D 01 46
End
```

**Figure 3 — Protocol Builder**

Your new protocol code is in the upper right corner ready for you to use. Finally, you should save this code by using the SAVE File... button, just in case you want to re-load it in the future to make changes. You should also post the saved file in the JP1 Forum file section.

Next step is creating an upgrade to use this new protocol, for that we need to switch over to KM.

## ***Using your new protocol in KM.***

In KM, you should select "Manual Settings" as the protocol and you should copy & paste the protocol code generated by PB into the Notes panel. (Be sure to use Edit, Paste Special, Values when pasting into the KM Notes panel.) You can still enter notes in the Notes panel too, if you like; KM will still find the protocol. On the Functions sheet, you should also select OBC entry mode for the button codes.

When you select "Manual Settings" a new set of options appears in the lower left hand side of the screen.

**PID:** this is the protocol ID that you used in PB, in our case 1FF.

**2<sup>nd</sup> Cmd Byte:** this is only relevant for signals with more than 1 byte of variable data. For this example, leave it set to "None".

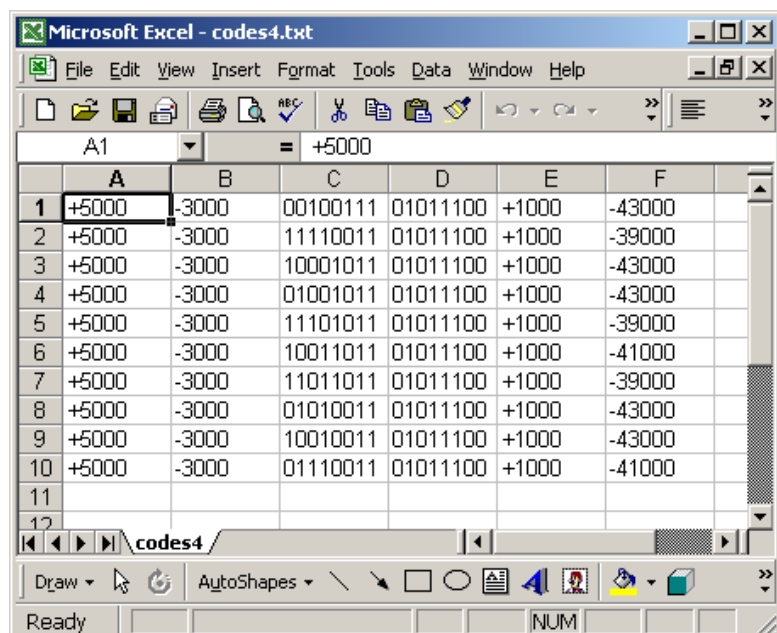
**Signal Style:** Sometimes this one is a crap-shoot. Take a look at the binary signals shown in the codes3.txt file. Ideally, you would notice that the numbers increase from the left (LSB) or the right (MSB) when you're looking at the data for the numeric buttons, but that's not exactly the case here. However, you should notice that the last 2 bits on the right stay constant whereas the bits on the left change constantly, therefore I'm going to make the call that this protocol is LSB. Use of one of the -COMP styles implies that you got the '1' and '0' pairs reversed, which is not typically the case for a protocol that you created yourself.

**Bits/Dev:** this should match what you selected in PB.

**Bits/Cmd:** this should match what you selected in PB.

So far, so good, but now you need to enter the functions and their OBCs in the Functions sheet, so you need to figure out what they are, and you still need to figure out what to enter as the device code for this upgrade.

codes4.txt is a cleaned up version of codes3.txt with just one line per signal. You should open this file using Excel. Select "fixed width" and make sure all the columns are treated as text. You should end up with a sheet that looks like this...



	A	B	C	D	E	F
1	+5000	-3000	00100111	01011100	+1000	-43000
2	+5000	-3000	11110011	01011100	+1000	-39000
3	+5000	-3000	10001011	01011100	+1000	-43000
4	+5000	-3000	01001011	01011100	+1000	-43000
5	+5000	-3000	11101011	01011100	+1000	-39000
6	+5000	-3000	10011011	01011100	+1000	-41000
7	+5000	-3000	11011011	01011100	+1000	-39000
8	+5000	-3000	01010011	01011100	+1000	-43000
9	+5000	-3000	10010011	01011100	+1000	-43000
10	+5000	-3000	01110011	01011100	+1000	-41000
11						
12						



The first two columns show the lead in times and the last two show the lead out times, so you can delete those. As we determined that the signal style is LSB, in order to convert the binary signals to decimal, we will need to read them backwards. The easiest way to reverse the order of binary patterns in Excel is to use the MID function. The following formula would reverse the order of an 8 bit binary pattern in cell A1:

```
= mid(a1,8,1)&mid(a1,7,1)&mid(a1,6,1)&mid(a1,5,1)&mid(a1,4,1)&mid(a1,3,1)&mid(a1,2,1)&mid(a1,1,1)
```

Assuming that the two columns of binary are now in columns A and B, you could enter this formula in column C and then copy it to column D, which would give you two columns of binary where the data from columns A and B are reversed.

You could then use the following formula in column E to convert the command codes to decimal:

```
=bin2dec(c1)
```

*NOTE: In most versions of Excel you will need to have the Analysis Toolpak loaded to use the BIN2DEC() function. If you do not have access to the Analysis Toolpak functions, an alternative formula may be used:*

```
=SUMPRODUCT(2^(8-ROW(INDIRECT("1:8")))*MID(RIGHT("0000000"&C1,8),ROW(INDIRECT("1:8")),1))
```

You could then copy this formula to column F in order for it to convert the device codes to decimal. You should end up with a sheet that looks like this (I added column headings to help illustrate the fields). (I have included this spreadsheet in the zip file; it's called codes4.xls)

	A	B	C	D	E	F	G
1	cmd-bin	dev-bin	cmd-bin2	dev-bin2	OBC	DEV	
2	00100111	01011100	11100100	00111010	228	58	
3	11110011	01011100	11001111	00111010	207	58	
4	10001011	01011100	11010001	00111010	209	58	
5	01001011	01011100	11010010	00111010	210	58	
6	11101011	01011100	11010111	00111010	215	58	
7	10011011	01011100	11011001	00111010	217	58	
8	11011011	01011100	11011011	00111010	219	58	
9	01010011	01011100	11001010	00111010	202	58	
10	10010011	01011100	11001001	00111010	201	58	
11	01110011	01011100	11001110	00111010	206	58	
12							

Therefore, the device code for our upgrade is 58, and the OBCs for the numeric buttons are as shown.

When entering the OBCs on KM's Functions sheet, remember to select OBC entry mode by clicking on the OBC button before starting. (The heading in column B should show OBC.)

The Setup sheet in KM should look like this...

Microsoft Excel - keymap-master-v9.14.xls

File Edit View Insert Format Tools Data Window Help

Tahoma 10 B U

sSetupCode = 2000

1 v9.14 JP1 Remote, Signature: RSL6RSL0

2 Remote: 15-1994

3 Device Type: Cable

4 Setup Code: 2000

5 Using Extender:

6

7

8 Protocol Name: Manual Settings

9 Device1: 58

10 Device2:

11 Device3:

12 Raw Fixed Data:

13

14

15

16

17

18

19 Protocol ID: 01 FF

20 Fixed Data: 5C

21 Manual Settings

22 PID: 1FF

23 2nd Cmd Byte: None

24 Signal Style: LSB

25 Bits/Dev: 8

26 Bits/Cmd: 8

27 Using 3rd-Party Protocol Code and PID from Line Notes

28 (see [Protocol Help] for details)

29

30

31

32

Upgrade Description:

Next Level

NEW LOAD SAVE UNDO Import Goto IR

Filename: (Untitled)

\*\*\* The Protocol Code below is REQUIRED for this Upgrade \*\*\*

Enter Device Codes and/or Raw Fixed Data (hex). Enter the Protocol ID and set the options in the Manual Settings table. Use the Notes section to enter any 3rd-Party Protocol Code.

15-1994 Device Upgrade Code: Buttons: 0 of 38 (4 bytes)

Upgrade Code 0 = 0F D0 (Cable/2000) Next Level (KM)

FF 00 01 5C

End

DISABLE Combined Key Moves DISABLE Embedded Notes

S3C8 Protocol Code (28 bytes):

Upgrade Protocol 0 = 01 FF (S3C8) Custom Protocol

3C 90 11 8B 14 F5 56 08 08 01 F4 05 C8 01 F4 01

E0 BF 68 09 C4 05 C8 FF 02 8D 01 33

End

Key Move Code:

Notes / 3rd-Party Protocol Code

This is for the Next Level cable

Upgrade protocol 0 = 01

3C 90 11 8B 14 F5 56 08

E0 BF 68 09 C4 05 C8 FF

End

Setup Functions Buttons Key Moves Protocol Help Protocols Layout Key Map Code List

Ready NUM

The Functions sheet in KM should look something like this:

Microsoft Excel - keymap-master-v9.14.xls

File Edit View Insert Format Tools Data Window Help

Tahoma 10 B U

B11 = 201

Functions	OBC	byte2	5-digit EFC	3-digit EFC	b2/ov	Hex	Description / Notes
num 0	206		00153	153		73	
num 1	228		00251	251		27	
num 2	207		00149	149		F3	
num 3	209		00086	086		8B	
num 4	210		00088	088		4B	
num 5	215		00085	085		EB	
num 6	217		00214	214		9B	
num 7	219		00212	212		DB	
num 8	202		00152	152		53	
num 9	201		00150	150		93	
vol up							
vol down							
mute							
channel up							
channel down							
power							
menu							
program/guide							
display/info							
exit							
up arrow							
down arrow							
left arrow							
right arrow							
select							
enter							
tv/vcr							
last/prev ch							
sleep							
play							

Clean Up

Notes:

Enter OBC-style Button Codes.

The default list of generic functions can be used as-is, overwritten, and/or deleted as desired.

Please do NOT confuse function names; they are separate.

Enter the function names from remote or device along with the OBC's for each of them.

If the byte2 column (C) is grayed, selected Protocol uses 1-byte code; the extra byte will be calculated automatically and no byte2 data entry is required.

If the byte2 column (C) is bright yellow, extra byte is required and must be entered.

If the byte2 column (C) is yellow, byte will be automatically calculated and overridden by entering a value.

Setup Functions Buttons Key Moves Protocol Help Protocols Layout Key Map Code List

Ready NUM