



Home Theater Solutions

HDS4x2

RS-232 Serial Protocol Reference

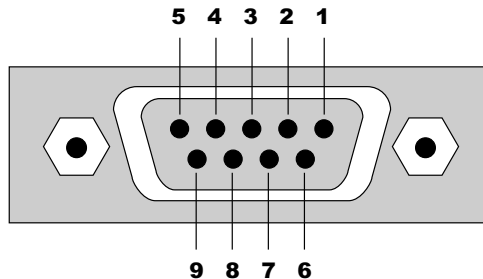
| | |
|---|----|
| RS-232 Port Hardware | 3 |
| RS-232 Pinout and Baudrate Settings | 3 |
| HDS4x2 Quick Reference | 4 |
| Command Syntax | 4 |
| Command Responses | 4 |
| The Acknowledgement Response | 4 |
| The Error Response | 4 |
| Query Strings | 5 |
| Checksums and CRC8's | 5 |
| HDS4x2 Quick Command Reference | 6 |
| Table of HDS4x2 Commands | 6 |
| A Simple Control Example | 7 |
| Powering on / off the HDS4x2 | 7 |
| Remapping an input to an output zone | 7 |
| Remapping an input to an output zone using the breakaway feature | 7 |
| Sequencing through inputs | 7 |
| Muting a zone | 8 |
| Reading the current settings | 8 |
| HDS4x2 Command Reference | 9 |
| The HDS4x2 Command Set | 9 |
| Error Response Codes | 9 |
| Command Definitions | 10 |
| '!' Resend Error Code | 10 |
| 'B' Button Emulation | 10 |
| 'BE' Button Enable / Disable | 11 |
| 'DZ' Delay Settings (*) | 12 |
| 'FS' Restore Factory Default Settings (*) | 13 |
| 'IR' Query Last IR Code Received (*) | 13 |
| 'IRE' Enable / Disable IR Remote Buttons | 14 |
| 'IRC' Set Learnable IR Code | 14 |
| 'LI' Front Panel Light Intensities and Mode (*) | 15 |
| 'MZ' Mute (*) | 16 |
| 'P' Power Control (*) | 17 |
| 'Q' Query Status (*) | 17 |
| 'QZ' Query for Zone Information (*) | 18 |
| 'SS' Save States (*) | 18 |
| 'SZ' Set Input (*) | 19 |
| 'V' Version Query (*) | 20 |
| 'XE' Transmit Enable Settings (*) | 20 |
| 'XS' Control Settings (*) | 21 |
| ' <i>ASY</i> ' Enable / Disable the Asynchronous Mode | 22 |
| ' <i>CSE</i> ' and ' <i>CRC</i> ' Checksum and CRC8 Options | 22 |
| ' <i>SET</i> ' Enable / Disable the Setup Mode | 23 |
| ' <i>BTE</i> ' Enable / Disable Front Panel Buttons | 23 |
| ' <i>IRE</i> ' Enable / Disable IR Buttons | 23 |
| ' <i>IRS</i> ' Enable / Disable Front Panel IR Sensor | 23 |
| ' <i>IRJ</i> ' Enable / Disable Rear Panel IR Jack | 23 |
| ' <i>AON</i> ' Enable / Disable Auto-on with Button Press Option | 23 |
| ' <i>12V</i> ' Enable / Disable 12V Control Option | 24 |
| ' <i>VMU</i> ' Enable / Disable Video Blanking when a Zone is Muted | 24 |
| ' <i>AMU</i> ' Enable / Disable Audio Muting when a Zone is Muted | 24 |
| K.I.S.S.™ Keep It Simple Serial | 25 |
| The K.I.S.S.™ Command Structure Overview | 25 |
| Using Bitmapped Parameters | 26 |
| Appending a Checksums or CRC-8 Checkcode to a Command | 26 |
| Clearing the Command Buffer | 27 |
| Response Strings | 27 |
| Response String Checksums and CRC-8 Checkcodes | 28 |
| Master / Slave and Asynchronous Modes of Operation | 29 |
| The Master / Slave Mode of Operation | 29 |
| The Asynchronous Mode of Operation | 29 |
| Calculating Checksums and CRC-8 Checkcodes | 30 |
| Differences between a Checksum and a CRC-8 Checkcode | 31 |
| Source Code Example of Calculating a Checksum | 32 |
| Source Code Example of Calculating a CRC-8 Checkcode | 33 |

RS-232 Pinout and Baudrate Settings

The RS-232 port on the HDS4x2 is the same format, and pinout, as a PC modem, and uses the same type of cable as a standard serial modem would, which is a standard straight through cable. Do not use a cable that is marked as a “Null Modem” cable.

The HDS4x2 can also be used with any USB to RS-232 conversion cable, these are all typically straight through cables. *(Be sure to install any drivers that come with the USB to RS-232 cable you are using.)*

The RS-232 port is a female type DE-9 connector (sometimes mistakenly referred to as a DB-9 connector) with the following pinout:



Pin definitions:

| | |
|----------------|----------------|
| 1 - No Connect | 6 - No Connect |
| 2 - TX | 7 - No Connect |
| 3 - RX | 8 - No Connect |
| 4 - No Connect | 9 - No Connect |
| 5 - GND | |

The port settings used by the HDS4x2 are:

Baudrate: 9600
Data Bits: 8
Stop Bits: 1
Parity: NONE

Command Syntax

The HDS4x2 serial command set uses an ASCII based protocol and a terminal emulator, like Hyperterm, can be used to test the serial port of the HDS4x2.

Each serial command is formatted as:

```
CMD param1,param2,...<CR>
```

Where:

CMD = The name of the command.

param = Any number of parameters can follow a command.

<CR> = The carriage return character, usually the character associated with the "Enter" key.

For instance the name of the command turn power on and off is 'P' (must be capitalized) therefore, to turn on the HDS4x2 send a:

```
P 1<CR>
```

to turn off the HDS4x2 send a:

```
P 0<CR>
```

the commands will not be echoed back, but instead you will receive a command response string.

Command Responses

The HDS4x2 will always respond to a command, there are no "time-out" modes, if you send a command and don't get a response in 100ms, something's wrong with the connection.

There are three different types of responses: Acknowledgements, Errors and Query Strings.

The Acknowledgement Response

Every command will get either an acknowledgement or error response.

Anytime you issue a command and there are no errors, you will receive the acknowledgement response.

Which is:

```
+<CR><LF>
```

which is the '+' character followed by a carriage return (hex=0Dh) and a line feed (hex=0Ah).

The Error Response

Every command will get either an acknowledgement or error response.

If something is wrong with the command, you will get an error response. Which is

```
!<err><CR><LF>
```

which is the '!' followed by an error number (in ASCII), followed by a carriage return and line feed.

For instance '2' is not allowed as a parameter in the 'P' (power) command, so:

```
P 2<CR>
```

would cause the HDS4x2 to respond with:

```
!2<CR><LF>
```

the '2' being the error code for "Parameter out of range".

For the full listing of error codes, see: ["Error Response Codes" on page 9](#)

Command Responses (Continued)

Query Strings

A query string is a command string that you would normally send to the HDS4x2, preceded by the '=' character, and with the parameters filled in to reflect the current state of the command. Commands return a query string when a '?' is used as a parameter.

For instance to find the current power on / off state of the HDS4x2 send:

```
P ?<CR>
```

and, assuming the power is on, it will respond with:

```
+<CR><LF>  
=P 1<CR><LF>
```

The '+' is the acknowledgement, indicating there were no errors in the command, the next line is the power command's current setting, the '1' indicates the power is turned on.

You can setup the HDS4x2 to send query strings when something changes, for instance when the power is toggled. See: ["Master / Slave and Asynchronous Modes of Operation" on page 29](#)

Checksums and CRC8's

You can optionally append a checksum or a CRC8 checkcode to the end of any serial command and it will be tested by the HDS4x2. If the checksum or CRC8 checkcode does not match, the command will be rejected with a "!4" error response.

To add a checksum to a command, append a ';' followed by the calculated checksum. For instance to turn on the HDS4x2 using a checksummed command:

```
P 1;220<CR>
```

To add a CRC8 checkcode to a command, append a ':' followed by the CRC8 checkcode. For instance to turn on the HDS4x2 using a CRC8 checkcoded command:

```
P 1:221<CR>
```

By setting the proper control flags, you can also have the HDS4x2 append a CRC8 checkcode or a checksum to all of its response strings. See: ["'XS' Control Settings \(*\)" on page 21](#).

For more information on checksums and CRC8 checkcodes, and how they're calculated, see: ["Calculating Checksums and CRC-8 Checkcodes" on page 30](#).

HDS4x2 Quick Command Reference

The quick reference tables and command descriptions refer to Inputs, Zones (or outputs) and Channels.

Inputs = The number of inputs a switch has. The HDS4x2 is a 4x2 switch and has 4 Inputs.

Zones = Number of Zones, or outputs, a switch has. The HDS4x2, a 4x2 switch, has 2 Zones.

Channels = The breakaway channels. These are signals that can be switched independently of each other. They are referred to in the commands by a single letter.

The HDS4x2 has 3 channels:

- Y - Component Video (Y/Pb/Pr).
- A - Analog audio (the L&R stereo connections).
- D - Digital audio.

Some notes on using the tables:

- Each command is followed by a <CR> (not shown in the tables).
- The 'Command' column shows the syntax of the commands. The values in italics are variables and are described in the 'Comments' column. The non-italic characters are literals and should be used as shown.
- Many commands allow a variable number of arguments, the examples shown here are common usages of the commands, for more details refer to the complete command reference.
- All command names are case sensitive. The parameters used in commands are not case sensitive.
- Most commands can be queried by replacing their parameters with a '?' character.

The commands in this table that are marked with a 'x' in the 'U' column are universal throughout most of the Zektor product line. By using only these commands a Zektor universal driver can be more easily written.

Table of HDS4x2 Commands

| U | Command | Description | Comments |
|---|----------------------------|---------------------------------|---|
| x | ! | Retransmit last error code | This command has no parameters, and causes the last error code to be retransmitted. |
| x | P <i>p</i> | Power control | <i>p</i> =power state (0=off, 1=on, +=toggle). |
| x | SZ <i>z,n</i> | Set zone to input | <i>z</i> =Zone (1-2), <i>n</i> =Input (1-4,+,-). |
| x | SZ <i>z,Yn,An,Dn,...</i> | Set zone to input w/breakaway | <i>z</i> =Zone (1-2), <i>n</i> =Inputs (1-4,+,-). Y=Y/Pb/Pr video, A=Stereo, D=Digital. |
| x | MZ <i>z,m</i> | Mute zone | Mutes all channels. <i>z</i> =Zone (1-2), <i>m</i> =Mute (0=Normal, 1=Muted, +=Toggle). |
| x | MZ <i>z,Ym,Am,Dm,...</i> | Mute zone w/breakaway | Mutes only given chans. <i>z</i> =Zone (1-2), <i>m</i> =Mute (0,1,+), Y=Y/Pb/Pr video, A=Stereo, D=Digital. |
| x | DZ <i>z,Yd,Ad,Dd</i> | Set zone switching delays | Set delay times chans. <i>z</i> =Zone (1-2), <i>d</i> =Delay (in ms). Y=Y/Pb/Pr video, A=Stereo, D=Digital. |
| x | V ? | Get version string | Returns: Product name and firmware version string. |
| x | LI <i>mode,dim,bri,off</i> | Sets LED intensities | <i>mode</i> =Mode (0=off, 1=dim, 2=bright, 3=auto), <i>dim,bri,off</i> =Dim, Bright and Off levels (0-100). |
| x | Q ? | Query for status | Returns: Operating status flags (see text). |
| x | QSZ ? | Query setting change bitmap | Returns: A bitmap of the zones that have had their input settings changed by the user. |
| x | QMZ ? | Query mute change bitmap | Returns: A bitmap of the zones that have had their mute settings changed by the user. |
| x | QZ ? | Query for no. of zones & inputs | Returns: Number of zones, number of inputs, and breakaway channel options. |
| x | IR ? | Read last IR code | Returns: Last IR code received. |
| | IRC <i>ircmd,ircode</i> | Set (learn) an IR code | <i>ircmd</i> =Command being set (see text), <i>ircode</i> =New IR code for command. |
| | IRE <i>ircmd,setting</i> | IR key enable/disable | <i>ircmd</i> =IR command to enable / disable, <i>setting</i> =EnableFlag (0=Disable,1=Enable) |
| | B <i>b,b,...</i> | Emulate button presses | <i>b</i> =Button Codes (up to 16 codes per 'B' command, see text). |
| | BE <i>btn,setting</i> | Button enable/disable | <i>btn</i> =Front panel button to enable / disable, <i>setting</i> =EnableFlag (0=Disable,1=Enable) |
| x | SS <i>n,\$</i> | Save current settings | Save current settings in EEPROM. <i>n</i> =Settings to save (see text). |
| x | FS 246,\$ | Reset to factory settings | Resets <i>everything</i> back to their factory default settings! |
| x | XS <i>flags</i> | Control settings | Set / reset control options. <i>flags</i> =Control option flags (see text). |
| x | XE <i>flags</i> | Transmit enable flags | Enable commands to asynchronously send their status. <i>flags</i> =Enable flags (see text). |

HDS4x2 Quick Reference (Continued)

A Simple Control Example

Here is an example of a few of commands that will most likely do everything you'll need in a simple installation.

<CR> = A Carriage Return, all commands end with a <CR>.
<CR><LF> = A Carriage Return, Line Feed, all responses end with a <CR><LF>.

If you are typing these commands manually, (using some sort of terminal program) it should be noted that the HDS4x2 does not echo your commands back as you type them, and all commands must be capitalized.

Powering on / off the HDS4x2

To power on the HDS4x2 issue the following command:

P 1<CR>

To power off the HDS4x2 issue this command:

P 0<CR>

Remapping an input to an output zone

To switch a zone to a new input:

SZ zone,input<CR>

Where:

zone = The zone you want to remap, a number: 1-2.
input = The input you want mapped to the given zone, a number: 1-4.

For instance if you want input zone 1 to display input 3:

SZ 1,3<CR>

Remapping an input to an output zone using the breakaway feature

To switch just the audio, or just the video channel of a zone, use the breakaway prefixes.

To switch just the analog audio on input 2 to zone 1:

SZ 1,A2<CR>

To switch the Y/Pb/Pr (component) video channel of input 3, and the digital audio channel of input 4 to zone 2:

SZ 2,Y3,D4<CR>

The by proceeding an input number with a channel character, only the channel of the input will be switched.

The channels used by the HDS4x2 are:

Y = Y/Pb/Pr (Component) video.
A = Analog audio.
D = Digital audio.

Sequencing through inputs

You can use the '+' or '-' characters in place of an input to sequence a zone through the inputs:

SZ 1,+<CR>

This will cause the input on zone 1 to be incremented.

SZ 2,A-<CR>

This will cause the analog audio channel on zone 2 to be decremented. The video and analog audio will remain unchanged.

A Simple Control Example (Continued)

Muting a zone

You can mute and unmute a zone by:

```
MZ zone ,mute<CR>
```

Where:

zone = The zone to be muted.

mute = The mute setting, 0=Unmuted, 1=Muted.

For instance to mute everything on zone 1:

```
MZ 1,1<CR>
```

To mute just the analog and digital channels on zone 2, and leave the video unchanged:

```
MZ 2,A1,D1<CR>
```

Reading the current settings

To query the HDS4x2 for its current state, use the '?' parameter, and the HDS4x2 will respond with a '=' followed by the command with the current settings as parameters.

To read the power on state of the HDS4x2

```
P ?<CR>
```

the HDS4x2 will respond with:

```
+<CR><LF>
```

This indicates the command was accepted with no errors

```
=P 1<CR><LF>
```

This indicates the HDS4x2 is currently ON

To read the mapping of a zone 1:

```
SZ 1,?<CR>
```

the HDS4x2 will respond with:

```
+<CR><LF>
```

This indicates the command was accepted with no errors

```
=SZ 1,2<CR><LF>
```

This indicates the zone 1 is currently mapped to input 2

To read the mapping of all zones with a single command:

```
SZ ?
```

the HDS4x2 will respond with a '=SZ' response for each zone.

The HDS4x2 Command Set

Examples are given for many of the ways a command can be issued. Most commands have a 'set' mode for changing a parameter value, and a 'query' mode for reading the current value.

If the command has a query mode, the associated response string, it will be listed.

As described in the section on the K.I.S.S.™ protocol, whitespaces and commas are optional in many cases. The format used here includes a single space after the command and commas, with no spaces, between parameters. The format given here does not show the optional checksum or CRC-8 checkcodes that may be appended to all commands, nor does it show the required <CR> that terminates all commands.

The response strings are the strings returned from HDS4x2, which use the same format as described above. The format does not show the optional checksum or CRC-8 checkcodes that may be appended to all response strings if enabled, nor does it show the <CR><LF> that terminates all Response Strings.

Error Response Codes

The following are the Error Response codes that can be returned by the HDS4x2:

- !1 Unrecognized command.
- !2 A parameter was out of range.
- !3 Syntax error, a badly formed command.
- !4 Checksum or CRC-8 error.
- !5 Too many or too few parameters.
- !6 Device busy, cannot process command.
- !7 Buffer overflow.
- !8 Command not valid if device is not powered on.

And some more detailed descriptions of their meanings:

Error 1: The command given was not recognized as a HDS4x2 command. Commands are case sensitive and in the HDS4x2, all commands are upper case.

Error 2: One of the parameters given was too large, or too small, the command will be ignored.

Error 3: Something was wrong with the command's syntax. There was possibly extra data at the end of the line, or non-decimal data as part of a parameter. There cannot be whitespace before or after a checksum or CRC-8 checkcode, or this error will be returned.

Error 4: The ';' or ':' character was used to indicate a Checksum or CRC-8 Checkcode was appended to the command string, but the Checksum or CRC-8 Checkcode did not match the calculated one. The command will be ignored.

Error 5: The number of parameters given does not match the number allowed by this command.

Error 6: To prevent conflicts between the front panel Setup Mode and the serial port settings, when the HDS4x2 is in the Setup Mode, many parameters become read only and any attempt at writing them will return Error 6. Issuing the "Button Emulation" command with button code '0' can be used to exit the Setup Mode, at which point the command can be re-issued without an Error 6 response.

Error 7: An internal buffer has overflowed, for instance more than 16 button codes were sent as part of the "Button Emulation" command.

Error 8: Power to the device must be 'ON' before this command is allowed.

Command Definitions

The commands are defined in alphabetical order.

(*) In an effort to make “Zektor Universal Drivers” easier to write, Zektor has adopted a subset of “Universal Commands”. By using only these commands, it should be possible to write a driver that works with all Zektor products. Reducing the effort of writing drivers for multiple Zektor products. The commands that are part of the Universal Command Set will be marked with a (*).

‘!’ Resend Error Code

This special purpose command is used to request that the HDS4x2 resend the last error code sent. This can be useful if the last error code sent had a checksum or CRC8 code appended to it that did not match.

! Request that the last error code sent, be resent

‘B’ Button Emulation

This command allows access to the internal keyboard handling of the HDS4x2, and is very hardware dependent. Button values returned by the HDS4x2 may and most likely will be different than button values returned by other Zektor devices.

Each button generates a value upon being pressed, and a different value upon release.

The Power toggle button also generates a unique value when held for 4 seconds, which is used to enter the setup mode. Other combinations may also generate unique codes.

This command allows the controller to detect front panel button presses even when the front panel is disabled. This allows the controller very tight control over the HDS4x2. By disabling the front panel (see: [“XS’ Control Settings \(*\)” on page 21](#)), and by then processing the front panel button presses of the HDS4x2, the user can redefine the operations of the HDS4x2.

When used in combination with the “Read Last IR Code” command, even IR commands can be handled by the user, outside the HDS4x2’s firmware.

Because of the tight link between this command and the HDS4x2’s firmware, there are some caveats when using this command. The Zektor firmware expects a button press code to always be followed by a button release code. Sending these codes out of logical order will not harm the HDS4x2, but may result in unpredictable behavior (buttons codes ignored, or unexpected state changes).

B *b1,bn...* Send one or more button codes to the HDS4x2.
B ? Query for any buffered button presses.
B Query for any buffered button presses.

Response String:

=B *b1,bn...*

Where:

b1,bn.. = A variable number of button codes (1 to 16 codes per command).

In the Master / Slave mode, only the last 16 button presses will be logged between queries, after that, new button presses overwrite the old ones in the internal buffer and will be lost to the controller.

The maximum number of button codes that can be sent is 16. If more than 16 button codes are sent a “parameter count error” will be returned and only the first 16 button codes will be accepted.

'B' Button Emulation (Continued)

The Button Codes for the HDS4x2's front panel are defined as follows:

| <u>Button</u> | <u>Press Code</u> | <u>Release Code</u> |
|---------------|-------------------|---------------------|
| Power Toggle | 1 | 9 |
| 1 | 2 | 10 |
| 2 | 3 | 11 |
| 3 | 4 | 12 |
| 4 | 5 | 13 |
| Breakaway | 6 | 14 |
| Zone 1 | 7 | 15 |
| Zone 2 | 8 | 16 |

The "Press Code" is the value returned when a button pressed, and the "Release Code" is the value returned when a button is released.

There are also quite a number of codes that are unique to this command that cannot be generated by pressing a front panel button. These extended codes allow for better control of the HDS4x2.

Extended button codes are:

| <u>Code</u> | <u>Description</u> |
|-------------|--|
| 0 | When Issued: Exits any setup modes. |
| 0 | When returned by query: No buttons have been pressed since last the query. |

Codes 100-147 map to the IR key codes 1-48, and behave the same as pressing an IR key.

Codes 148-158 map to the extended codes 49-59, and behave as if an extended IR code were received.

The '0' code has special meaning. When returned in a Query Response string it means there are no keys waiting in the buffer. When issue by the user, it acts like an exit key, used to exit setup modes, similar to pressing the Power Toggle button, but it will be ignored if the HDS4x2 is not in a setup mode. By issuing '0' codes, the HDS4x2 can be returned to a known state, regardless of any possible setup state it might be in.

The '0' button code is also device independent. Its use, and value, does not change between Zektor devices like the other codes may (and most likely will).

The HDS4x2 responds to codes 100-147 as if the IR codes 1-48 were being received from the ZRM2 remote. The HDS4x2 responds to codes 148-158 as if the extended IR codes 49-59 were being received.

See: "['IRC' Set Learnable IR Code](#)" on page 14 for a list, and meaning of the IR and extended IR codes.

'BE' Button Enable / Disable

By using the 'XS' command, the entire front panel can be enabled/disable as a whole (see: "['XS' Control Settings \(*\)](#)" on page 21). However it is also possible to enable/disable subset of front panel buttons.

Once the front panel is disable by using the 'XS' command, individual buttons can be re-enabled.

For instance, if the breakaway option is not going to be used, a common example is to enable all buttons except the breakaway button.

The command format is:

| | |
|-------------------------------|---------------------------------------|
| BE <i>btn, setting</i> | Enable / disable a front panel button |
| BE <i>btn, ?</i> | Read the state of the 'btn' button |
| BE <i>?</i> | Read the state of all buttons |

Response String:

BE *btn, setting*

'BE' Button Enable/Disable (Continued)

Where:

btn = The button to be enabled / disable.

setting = The enable / disable setting. 0=Disable button, 1= Enable button.

Where '*btn*' maps to:

- 1 = Power toggle button.
- 2 = '1' button.
- 3 = '2' button.
- 4 = '3' button.
- 5 = '4' button.
- 6 = Video / audio breakaway button.
- 7 = Zone 1 button.
- 8 = Zonn 2 button.

An example of this command might be to disable the breakaway button. To do this (this example does not display the '+' responses that are returned by each command):

```
XS -2      Allow individual control of each button
BE 1,1     Re-enable all buttons but the breakaway
BE 2,1
BE 3,1
BE 4,1
BE 5,1
BE 6,0     Make sure the breakaway button is disabled
BE 7,1
BE 8,1
```

This command has two ways to query its settings. This first:

```
BE btn,?
```

will return the current setting of just the '*btn*' button.

Whereas:

```
BE ?
```

will return all 8 button settings as individual query response strings, it has a similar affect as issuing individual 'BE *btn*,?' commands for each button.

'DZ' Delay Settings (*)

Allows setting of the switching delays. Switching delays refer to the amount of time the channel is muted when switching from one input to another. This is sometimes called "Fade to Black" even though there is no "Fading" involved. When switching from one channel to another, the HDS4x2 will mute the channel for a programmable amount of time. For instance if the digital audio section is set to 200ms, then anytime the digital audio section is switched, the digital audio will be muted for 200ms before being switched to the new channel. Each of the channels operate independently and different times can be set for each of the channels.

Some receivers can "thump" if their digital audio is switched too fast, switching delays can help prevent this.

By setting a video switching delay that's long enough to allow the monitor to detect a signal loss, it is sometimes possible to prevent the "rolling" that can occur when switching component video. Instead of rolling, the picture will go black, and re-appear displaying the new channels video, hence the term "Face to Black" that is sometimes used to describe a muting delay when switching.

'DZ' Delay Settings (Continued)

Because different monitor need different delay times to prevent rolling, the HDS4x2 allows different times to be set for each zone. And because the switching times needed for a monitor are probably different than those needed for digital or analog audio, each channel of each zone can have its switching times set independently.

DZ zone,Mnn,Ann,Dnn,Hnn,\$ Set new settings
DZ ? Query for current settings

Response String:

=DZ zone,Mnn,Ann,Dnn,Hnn

Where:

zone = Zone (1-2).
Ynn = Y/Pb/Pr switching delay in milliseconds (*nn*=milliseconds).
Ann = Analog audio switching delay in milliseconds (*nn*=milliseconds).
Dnn = Digital audio switching delay in milliseconds (*nn*=milliseconds).
\$ = Optional, but if present settings are backed up in EEPROM.

The 'Y', 'A', and 'D' characters are literals and are used to indicate which delay is being set. If no prefix is used, all channels of the given zone be set to the same delay.

The '*nn*' is a variable and indicates the delay time in milliseconds.

Not all parameters have to be present, and they do not have to appear in any particular order. For instance to set only the digital audio delay on zone 2 to 200ms:

DZ 2,D200

To set the analog audio (the L&R inputs) to use a 100ms delay, and the multichannel audio (the 5.1 audio) to use a 200ms delay, on zone 1:

DZ 1,A100,M200

'FS' Restore Factory Default Settings (*)

To restore the HDS4x2 to its factory default settings:

FS 246\$

This will set **everything** back to the programmed factory settings. This includes timings, IR codes (all learned IR codes will be lost), front panel LED intensity settings, etc. **Everything**, means everything!

'IR' Query Last IR Code Received (*)

This command allows the user to read the values returned by Zektor's IIR™ (Intelligent Infra-Red) decoding firmware. Zektor's IIR™ algorithm converts all IR codes it receives to a compressed, 72 bit value.

Each different key press of a remote control will generate a different but repeatable pattern.

This command returns a value for every IR code detected by the front panel IR sensor (or IR jack if enabled), regardless as to whether the IR code detected was used to control the HDS4x2.

The uses for this command are two fold:

1. The value returned from this command are the same values used to teach the HDS4x2 new IR codes over the serial port. See: ["IRC' Set Learnable IR Code" on page 14.](#)
2. This command gives the controller full access to the HDS4x2's IR sensor and Zektor's IIR™ algorithm. This is a very reliable way of adding IR control to any project. The IR codes generated by Zektor's IIR™ algorithm are immune to timing differences between universal remote control manufacturers, and to the timing errors associated with a low battery charge in the remote control.

'IR' Query Last IR Code Received (Continued)

Note 1: The Zektor's IIR™ algorithm works with any remote control code that is pulse width modulated. This is pretty much every type of IR code except the Phillips RC-5, and RC-6 codes.

Note 2: Because very few control systems could handle a 72 bit decimal value, and in an effort to keep the size of the IR response small, this command sends the 72 bit IR code as an 18 digit hexadecimal value.

The format of the command is:

IR ? Query for the IR code of the last IR command received.

Response String:

=IR *ircode*

Where:

ircode = 18 hex digits (0-9, A-F), representing the most recent IR code received.

This command returns a single digit '0' if there are no IR codes waiting to be read.

'IRE' Enable / Disable IR Remote Buttons

Using this command, combined with the 'IRE' setting in the 'XS' command (see: ["XS' Control Settings \(*\)" on page 21](#)), you can enable or disable individual commands received through the IR sensor.

This command has no affect on IR codes that are received through the rear panel IR jack. This allows you to keep the user from using buttons on the IR remote, and still be able to use the same commands to control the HDS4x2 through the IR jack.

IRE *ircmd, n* Enable or disable an IR command
IRE *ircmd, ?* Read the enable/disable state of '*ircmd*'
IRE ? Read the enable/disable states of all '*ircmd*'s

Response String:

=IRE *ircmd, n*

Where:

ircmd = IR command code.

n = Enable / disable setting. 0=Disabled, 1=Enabled.

The '*ircmd*' variable is the same as that used by the 'IRC' command, for a full listing of all the IR commands that can be enabled / disabled, see: ["IRC' Set Learnable IR Code" on page 14](#).

'IRC' Set Learnable IR Code

This command is used to set, or retrieve, the current IR codes associated with the learnable IR commands. This could be useful for "cloning" the IR codes learned in one HDS4x2 into another HDS4x2.

IRC *ircmd, ircode* Set the '*ircmd*' to use the IR code '*ircode*'.
IRC *ircmd, ?* Query for a single IR command.
IRC ? Query for all '*ircmd*' settings.

Response String:

=IRC *ircmd, ircode*

Where:

ircmd = IR command number being set / retrieved (See Table).

















































ircode = 72 bit (18 hex digits) IR code.

'IRC' Set Learnable IR Code (Continued)

The value '*ircmd*' refers to the IR commands that the HDS4x2 is able to learn.

The '*ircmd*' codes 1-48, are the IR codes for the buttons on the ZRM2.

This table shows the mapping of the '*ircmd*' and corresponding ZRM2 button:

| <i>ircmd</i> | ZRM2 | <i>ircmd</i> | ZRM2 | <i>ircmd</i> | ZRM2 | <i>ircmd</i> | ZRM2 |
|--------------|---|--------------|---|--------------|---|--------------|---|
| 1 |  | 2 |  | 3 |  | 4 |  |
| 5 |  | 6 |  | 7 |  | 8 |  |
| 9 |  | 10 |  | 11 |  | 12 |  |
| 13 |  | 14 |  | 15 |  | 16 |  |
| 17 |  | 18 |  | 18 |  | 19 |  |
| 21 |  | 22 |  | 23 |  | 24 |  |
| 25 |  | 26 |  | 27 |  | 28 |  |
| 29 |  | 30 |  | 31 |  | 32 |  |
| 33 |  | 34 |  | 33 |  | 34 |  |
| 37 |  | 38 |  | 39 |  | 40 |  |
| 41 |  | 42 |  | 43 |  | 44 |  |
| 45 |  | 46 |  | 47 |  | 48 |  |

The '*ircmd*' codes 49-59 are extended codes. These are IR codes that the HDS4x2 is capable of receiving, but are not available on the ZRM2. The HDS4x2 is also able re-learn these codes:

- 49 = A/V Toggle function.
- 50 = Sequence selected zones through inputs.
- 51 = Zone 1 Mute.
- 52 = Zone 1 Unmute.
- 53 = Zone 1 Mute Toggle.
- 54 = Zone 2 Mute.
- 55 = Zone 2 Unmute.
- 56 = Zone 2 Mute Toggle.
- 57 = Selected Zones Mute.
- 58 = Selected Zones Unmute.
- 59 = Selected Zones Mute Toggle.

Setting an '*ircmd*' to '*ircode* = 0', causes that command to no longer respond to IR, this includes the IR jack!

The proper way to disable an IR command, so that a button on the remote is ignored, yet the command is still available to the IR jack, is to use the 'IRE' command, see: ["IRE' Enable / Disable IR Remote Buttons" on page 14.](#)

Learned IR codes are always saved in EEPROM.

'LI' Front Panel Light Intensities and Mode (*)

Set the mode, dim and bright levels of the front panel LEDs:

- | | |
|---|--|
| LI <i>mode, dim, bright, off</i> | Set the MODE, DIM, BRIGHT and OFF levels. |
| LI <i>mode, dim, bright, off, \$</i> | Set levels, and backup settings in EEPROM. |
| LI ? | Query for current settings. |

Response String:

=**LI** *mode, dim, bright, off*

'LI' Front Panel Light Intensities and Mode (Continued)

Where:

- mode* = MODE setting.
- dim* = DIM level setting.
- bright* = BRIGHT level settings.
- off* = OFF level (level of LED intensity when in standby mode).

Where '*mode*' settings are:

- 0 = Turn off front panel lights.
- 1 = Front panel lights are always at *dim* level.
- 2 = Front panel lights are always at *bright* level.
- 3 = Front panel lights auto-dim after 4 seconds.

The *dim* and *bright* intensities range from 0=Off, to 100=Maximum brightness.

If the levels are set, but the '\$' suffix is not used, the levels the new levels will not be restored after a power failure. The '\$' suffix is used to backup the new settings in EEPROM, which will survive a power failure.

'MZ' Mute (*)

This command is used to mute the outputs of the HDS4x2, either all at once, or just selected channels.

To mute or unmuted outputs:

| | |
|---------------------------------------|----------------------------------|
| MZ <i>zone, m</i> | Mute or unmute everything |
| MZ <i>zone, Mm, Am, Dm, Hm</i> | Mute or unmute selected channels |
| MZ <i>zone, ?</i> | Read the state of a single zone |
| MZ <i>?</i> | Read the state of all zones |

Response String:

=**MZ** *zone, Mm, Am, Dm, Hm*

Where:

- zone* = Zone to be muted (1-2).
- m* = Mute value, 0=Normal (unmute), 1=Mute.
- Ym* = Mute or unmute the Y/Pb/Pr (component) video channel.
- Am* = Mute or unmute the analog audio channel.
- Dm* = Mute or unmute the digital audio channel.

The 'Y', 'A', and 'D' characters are literals and are used to indicate which channel is being muted. If no prefix is used, all channels will be affected.

The 'm' is a variable and indicates the mute setting (0=Not muted, 1=Muted).

Not all parameters have to be present, and they do not have to appear in any particular order. For instance to mute only the digital audio on zone 1:

MZ 1, D1

To mute the analog audio (the L&R inputs) and unmute the video on zone 2:

MZ 2, A1, Y0

'P' Power Control (*)

Turn on / off, or toggle the power state of the HDS4x2:

- P 0** Turn off power
- P 1** Turn on power
- P +** Toggle power
- P ?** Query for current setting

Response String:

=P n

Where:

n = Current power status, 0=Off, 1=On.

'Q' Query Status (*)

In the Master / Slave mode of operation, (see: ["The Master / Slave Mode of Operation" on page 29](#)), this command is used to poll for any pending state changes that are waiting to be read. By issuing this command and testing the returned bitmapped value, the controller can determine what has changed in the HDS4x2 since the last time it was polled.

This command allows the controller to quickly poll the HDS4x2, using only one command, instead of issuing a string of commands to check if the power state has changed, or if a new input has been selected, a button pressed, etc. The Query Status command is used to determine if anything has changed, and then based on the results of the Query Status, only the query commands needed are issued to read the new states of the HDS4x2.

Once the new state is read by issuing the proper query command, the associated flag will be reset.

- Q ?** Query for current flag values

Response String:

=Q flags

Where '*flags*' is a bitmapped parameter:

| Decimal Value | +128 | +64 | +32 | +16 | +8 | +4 | +2 | +1 |
|---------------|------|-----|-----|-----|-----|-----|-----|-----|
| Bit Position | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | SMC | 0 | 0 | MUT | IRR | BTN | SEL | PWR |

0 - Reserved, always returns 0.

PWR- 1=Power State has changed.

SEL - 1=Selection (Input / Output Mapping) has changed.

BTN - 1=One or more buttons have been pressed.

IRR - 1=A new IR code has been received.

MUT- 1=Mute settings have changed.

SMC- 1=Setup Mode Changes -- something has been changed using the setup mode.

This command uses a bitmapped parameter. See: ["Using Bitmapped Parameters" on page 26](#).

The 'SMC' bit will be set if any front panel control settings have changed, and will remain set until the changes made have been read using the appropriate query. The queries that affect the 'SMC' flag are:

- LI ?** The lighting mode or intensities have changed
- XS ?** The control setting options have changed
- BE ?** The button enable mask has changed
- IRE ?** The IR enable mask has changed

'QZ' Query for Zone Information (*)

Returns the number of Inputs, Zones, and Breakaway channels.

QZ ? Query for number of Inputs, Zones, and Breakaway channels

Response String:

=QZ *inputs,zones,breakaway_opt_1,breakaway_opt_n,...*

Where:

inputs = Number of inputs.

zones = Number of zones.

breakaway_opt_x = A variable number of breakaway options.

This command is used to determine the number of inputs, zones, and the different breakaway options available to a device.

Its response string is static (it always returns the same string).

For a HDS4x2, the response string will always read:

=QZ 4,2,Y,D,A

Which indicates 4 Inputs, 2 Zones, and the Breakaway options Y,D,A.

Where:

Y = Y/Pb/Pr (Component) video

D = Digital audio

A = Analog audio

'SS' Save States (*)

Backs up the current states of the HDS4x2 in EEPROM.

EEPROM is a type of memory that retains its data even when power is disconnected from the HDS4x2.

There are two types of data that this command can save in EEPROM.

The first is that data that is associated with all commands that can have a '\$' suffix added to them.

The second option is the power on state. By default, when power is first applied to the HDS4x2, it power on in the standby mode. If the power button it pressed, it powers itself on, with all channels pointing to input 1. This default behavior can be changed using this command.

Command format:

SS states,\$ Backup the selected states into EEPROM

Where:

states= States to backup.

1 = Backup the current configuration as the new power on settings.

2 = Backup all unsaved data from commands that can be suffixed with a '\$'.

3 = Backup all unsaved data and save new power on settings.

This command can be use to delay the saving of data in commands that can have their data saved in EEPROM.

For instance the LED mode settings can be changed using:

LI 3

This command will set the HDS4x2 to have its LED on bright all the time, but because the command was not suffixed with the '\$' character, the setting is only valid until power is disconnected from the HDS4x2. Once power is restored the settings used will be those that were previously saved in EEPROM.

'SS' Save States (Continued)

To save the new LI settings you can issue the LI command with just a '\$'

```
LI $
```

Or you can use the 'SS' command to save all unsaved data, which will include the LI setting:

```
SS 2,$
```

The 'SS' command is also used to change the initial power on settings.

For instance if when initially plugged in, you want the HDS4x2 to turn itself on and switch to input 2, you can:

```
P 1      Turn on the HDS4x2
SZ 1,2   Switch zone 1 to input 2
SS 1,$   Save the current configuration as the new power on settings
```

'SZ' Set Input (*)

This is the command used to map inputs to zones.

```
SZ zone,in           Switch all channels to new input
SZ zone,Yin,Ain,Din  Switch the breakaway channels' inputs
SZ ?                 Read the current settings
```

Response String:

```
=SZ zone,in           or,
=SZ zone,Yin,Ain,Din
```

Where:

```
zone = Zone 1-2.
in    = Input 1-4.
Y     = Y/Pb/Pr (component) video channel.
A     = Analog audio (L & R stereo) channel.
D     = Digital audio channel.
```

The command has two modes of operation, a non-breakaway and a breakaway mode.

If no breakaway channels are specified then it's assumed all channels are to be switched, for instance:

```
SZ 1,3
```

will set everything on input 3, to zone 1.

Whereas:

```
SZ 2,A3,D1,Y2
```

will switch the analog audio 'A' on input 3, the digital audio 'D' on input 1 and the component video 'Y' on input 2, to zone 2.

When a response string is requested:

```
SZ ?
```

The response will depend upon the breakaway settings, if all of them are the same (like the first example) then only the input will be returned. For instance the first example would return:

```
=SZ 1,3
```

If breakaway is in effect (different channels point to different inputs), then they will be listed separately. For instance, assuming the Multichannel audio is currently switched to '2', the second example would return:

```
=SZ 2,M2,A3,D1,Y2
```

'V' Version Query (*)

Query for the current firmware version of the HDS4x2.

v ? Request version string
v Request version string

Response String:

=v HDS4x2 firmware_ver firmware_serial_number

Where:

firmware_ver = Version number of the HDS4x2's firmware.
firmware_serial_number = Internal serial number (does not match the product's serial number)

'XE' Transmit Enable Settings (*)

In the Asynchronous mode of operation, the HDS4x2 will transmit state changes as they occur. This command allows individual control over which state changes will be transmitted.

This HDS4x2 allows asynchronous transmit control over the following states:

- Power State changes (On / Off condition).
- Selection changes (input / output mapping changes).
- Front Panel Button Presses.
- IR codes received.
- Changes in the mute status.

Each of the above states can be selectively set to asynchronously transmit their state changes, or run in the Master / Slave mode. If asynchronous transmit has been disabled for one of the options, then that option will revert to the Master / Slave mode.

The 'ASY' bit in the "Control Settings" command, (see: ["XS' Control Settings \(*\)" on page 21](#)), must be set to '1' to allow any Asynchronous transmissions, if the 'ASY' bit is set to '0', then the HDS4x2 will operate solely in the Master / Slave mode regardless of the settings of this command.

XE settings Set the enable bits to 'settings'
XE +settings Set enable bits indicated in 'settings' to 1
XE -settings Reset enable bits indicated in 'settings' to 0
XE settings, \$ Set the control bits to 'settings', save in EEPROM
XE ? Query for current settings

Response String:

=XE settings

Where 'settings' is a bitmapped parameter:

| Decimal Value | +16 | +8 | +4 | +2 | +1 |
|---------------|-----|-----|-----|-----|-----|
| Bit Position | 4 | 3 | 2 | 1 | 0 |
| Name | MUT | IRR | BTN | SEL | PWR |

Factory Settings 0 0 0 1 1

PWR- 1=Power State has changed.
 SEL - 1=Selection (Input / Output Mapping) has changed.
 BTN - 1=One or more buttons have been pressed.
 IRR - 1=A new IR code has been received.
 MUT- 1=Mute settings have changed.

This command uses a bitmapped parameter. Each bit can set or reset without affecting the other bits. See: ["Using Bitmapped Parameters" on page 26](#), for more information on using bitmapped parameters.

'XE' Transmit Enable Settings (Continued)

Each bit controls the transmission of a response string, when the 'ASY' bit in the 'XS' command is set and the associated state changes.

The response strings are:

- PWR - The '=P *n*' response string.
- SEL - The '=SZ *zone,in*' response string.
- BTN - The '=B *n*' response string.
- IRR - The '=IR *n*' response string.
- MUT - The '=MZ *zone,m*' response string.

If the bit in '*settings*' is set, and the 'ASY' bit in the 'XS' command is set, then the respective response string will be sent anytime its state changes, for a more detailed explanation see: ["XS' Control Settings \(*\)" on page 21](#).

The '\$' parameter is a "Backup Control Settings" flag. If it exists, then the current settings will be backed up in EEPROM, and will remain unchanged through a power failure. Backing up the "Transmit Enable Settings" will also backup the "Control Settings", see: ["XS' Control Settings \(*\)" on page 21](#).

'XS' Control Settings (*)

Turn on and off operational modes of the HDS4x2.

This command allows control over the following:

- Select the Master / Slave or Asynchronous modes of operations.
- Enable / Disable appending Checksums or CRC-8's to responses.
- Enable / Disable the front panel switches.
- Enable / Disable the IR control.
- Enable / Disable the IR jack.
- Enable / Disable auto power on with button press.
- Enable / Disable using the IR jack as a 12V control line.
- Enable / Disable video muting, when a zone is muted.
- Enable / Disable audio muting, when a zone is muted.
- Enable / Disable the setup mode.

The format of the command is:

| | |
|------------------------------------|---|
| xs <i>settings</i> | Set the control bits to ' <i>settings</i> ' |
| xs + <i>settings</i> | Set bits indicated in ' <i>settings</i> ' to 1 |
| xs - <i>settings</i> | Reset bits indicated in ' <i>settings</i> ' to 0 |
| xs <i>settings</i> , \$ | Set the control bits to ' <i>settings</i> ', and save in EEPROM |
| xs ? | Query for current settings |

Response String:

=**xs** *settings*

'XS' Control Settings (Continued)

Where 'settings' is a bitmapped parameter:

| Decimal Value | +2048 | +1024 | +512 | +256 | +128 | +64 | +32 | +16 | +8 | +4 | +2 | +1 |
|------------------|-------|-------|------|------|------|-----|-----|-----|-----|-----|-----|-----|
| Bit Position | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | AMU | VMU | 12V | AON | IRJ | IRS | IRE | BTE | SET | CRC | CSE | ASY |
| Factory Settings | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

| | |
|---|--|
| ASY - 0=Master / Slave mode. | 1=Asynchronous Mode. |
| CSE - 0=Disable the sending of CS's or CRC-8's. | 1=Append either Checksums or CRC-8 to responses. |
| CRC - 0=Append Checksums to responses if 'CSE' set. | 1=Append CRC-8's, to responses if 'CSE' is set. |
| SET - 0=Disable the setup option. | 1=Enable the setup option. |
| BTE - 0=Use 'BE settings for front panels buttons. | 1=Enable all front panel buttons, overrides 'BE' settings. |
| IRE - 0=User 'IRE' settings for IR control. | 1=Enable all IR commands, overrides 'IRE' settings. |
| IRS - 0=Disable IR sensor. | 1=Enable IR sensor. |
| IRJ - 0=Disable IR jack. | 1=Enable IR jack. |
| AON - 0=Disable auto-on with any button press. | 1=Enable auto-power on with button press. |
| 12V - 0=Disable 12V control. | 1=Enable 12V control. |
| VMU - 0=Disable video muting when a zone is muted. | 1=Enable video muting when a zone is muted. |
| AMU - 0=Disable audio muting when a zone is muted. | 1=Enable audio muting when a zone is muted. |

This command uses a bitmapped parameter. Each bit can set or reset without affecting the other bits. See: ["Using Bitmapped Parameters" on page 26](#), for more information on using bitmapped parameters.

If the '\$' parameter exists, then the current settings will be backed up in EEPROM, and will remain unchanged through a power failure. Backing up the "Control Settings" will also backup the "Transmit Enable Settings", see: ["XE' Transmit Enable Settings \(*\)" on page 20](#).

The following paragraphs define each option in more detail:

'ASY' Enable / Disable the Asynchronous Mode

By setting this bit the HDS4x2 will enter the asynchronous mode, which simply means each time a state changes, like the power being toggled, the HDS4x2 will send a response string immediately instead of waiting for a query request. This command works with the 'XE' command, when the 'ASY' option is enabled, the 'XE' settings determine which states will be immediately transmitted, and which will require polling. For a list of the available states see: ["XE' Transmit Enable Settings \(*\)" on page 20](#).

For instance, using the power toggle example, if you'd like for the HDS4x2 to transmit the '=P n' response each time the power on the HDS4x2 changes state:

```
XS +1 Set the 'ASY' option without affecting any other options
XE 1 Enable power toggle transmissions, while disabling everything else
```

Now each time the HDS4x2's power is turn on or off, the power response string will be sent:

```
=P n
```

where the 'n' will be a '0' or a '1', indicating the new power state.

To have each IR command sent as it's received, and to be informed of changes in the input to zone mappings, enable those options as well:

```
XE +10 Enable just the IRR and SEL options without affecting the others
```

Now each time a zone is remapped to an input, or an IR code is received, or the power state changes, you receive the '=IR n' and '=SZ zone,in' and '=P n' response strings will be sent.

'CSE' and 'CRC' Checksum and CRC8 Options

When the 'CSE' option is enabled, all response strings will have either a checksum, or a CRC8 checkcode appended to them. The 'CRC' option determines whether a checksum or CRC8 checkcodes is used. For a description of using checksums and CRC8 checkcodes see: ["Checksums and CRC8's" on page 5](#).

'XS' Control Settings (Continued)***'SET' Enable / Disable the Setup Mode***

This bit in 'settings' can be used to disable the ability to enter the setup mode, preventing a casual user from enter the setup mode and making changes, or reassigning presets.

Setting this bit to a '0' prevents the HDS4x2 from entering the setup mode when holding the Power Toggle button, it also disables the 'SET' button on the remote control.

It also disables the ability to perform a full factory reset by pressing and holding the Power Toggle, '2' and '3' buttons for 4 seconds.

However you can enable / disable the setup mode through the front panel by pressing and holding the Power Toggle, '1' and '2' buttons for 4 seconds (until the power toggles). This toggles between enabling and disabling the setup mode. This allows the installer to re-enable the setup mode, make front panel changes, and then disable the setup mode. When toggling between the setup mode being enabled, and disabled, the only indication as to whether the mode is enabled or not, is the ability to enter the setup mode.

'BTE' Enable / Disable Front Panel Buttons

Setting the 'BTE' to '0' allows each button on the front panel to be enabled or disabled by using the 'BE' command. See: ["*'BE' Button Enable / Disable*" on page 11](#). When the 'BTE' is set to '1', it will override all 'BE' settings, and all buttons will be enabled.

'IRE' Enable / Disable IR Buttons

Setting the 'IRE' to '0' allows each button on the remote control to be enabled or disabled by using the 'IRE' command. See: ["*'IRE' Enable / Disable IR Remote Buttons*" on page 14](#). When the 'IRE' is set to '1', it will override all 'IRE' settings, and all remote control buttons will be enabled.

'IRS' Enable / Disable Front Panel IR Sensor

Setting the 'IRS' bit to '0', disables the IR sensor on the front panel. This has no affect on IR signals being received through the rear panel IR jack. This is useful in preventing stray IR from affecting the HDS4x2.

'IRJ' Enable / Disalbe Rear Panel IR Jack

Setting the 'IRJ' bit to '0', disables the IR jack on the rear panel. This has no affect on IR signals being received through the front panel IR sensor. The best way to disable the IR jack is to simply unplug whatever it is you have plugged into it. However if you plan on using the IR jack as a 12V control line, where 12V turns the HDS4x2 on, and 0V turns it off, this option must be set to '0'. You cannot receive IR signals through the IR jack, and also use the IR jack as a 12V control line.

'AON' Enable / Disable Auto-on with Button Press Option

By default, pressing a button on the HDS4x2's front panel, or remote control, will turn on the HDS4x2.

This may not be desirable, or even counterintuitive to some. It is easily changed by setting the 'AON' bit to '0'.

When the 'AON' bit is set to '0', only by pressing the Power Toggle button on the HDS4x2, or pressing the Power Toggle button, or ON button on the remote will turn on the HDS4x2.

However the remote's preset buttons will still power on the HDS4x2. The presets are considered discrete power on commands, combined with their ability to map inputs to zones.

'XS' Control Settings (Continued)***'12V' Enable / Disable 12V Control Option***

A seldom used, but sometimes useful option is to use a 12V trigger voltage to turn on and off the HDS4x2. By setting the 'IRJ' bit to '0', and the '12V' bit to '1', the HDS4x2 will turn itself on whenever it detects a 0V to 12V transition, and turn itself off whenever it detects a 12V to 0V transition on the IR jack input.

The '12V' option and the 'IRJ' option are mutually exclusive, if you try to enable both of them at the same time, the '12V' will be set to '0'.

To turn on the '12V' option, you must first turn off the 'IRJ' option, using two commands:

```
xs -128      Turn off the 'IRJ' option
xs +512      Turn on the '12V' option
```

Any attempt to turn on the '12V' option while the 'IRJ' option is enabled, will be ignored.

'VMU' Enable / Disable Video Blanking when a Zone is Muted

When the HDS4x2 receives an extended IR commands to mute a zone, the 'VMU' setting determines if the video is blanked when the zone is muted.

If the 'VMU' bit is '0', then the video will be unaffected when an IR mute command is received.

If the 'VMU' bit is '1', then the video will be blanked when an IR mute command is received.

Setting both the 'VMU' and 'AMU' bits to '0', effectively disables the IR mute option.

This option has no affect on the muting that is done as part of the switching delays given in the 'DZ' command, it also has no affect on the 'MZ' command.

'AMU' Enable / Disable Audio Muting when a Zone is Muted

When the HDS4x2 receives an extended IR commands to mute a zone, the 'AMU' setting determines if the audio is muted when the zone is muted.

If the 'AMU' bit is '0', then the audio will be unaffected when an IR mute command is received.

If the 'AMU' bit is '1', then the audio will be muted when an IR mute command is received.

Setting both the 'VMU' and 'AMU' bits to '0', effectively disables the IR mute option.

This option has no affect on the muting that is done as part of the switching delays given in the 'DZ' command, it also has no affect on the 'MZ' command.

The K.I.S.S.™ Command Structure Overview

The section of the manual describes the overall structure of the K.I.S.S.™ protocol. It only needs to be read if you are having some problems with the way a product is behaving, or you just like reading this sort of thing.

The following conventions are used to describe the protocol:

| | |
|------------|---|
| <CR> | = An ASCII Carriage Return ('0D' hex) |
| <LF> | = Line Feed ('0A' hex) |
| <ESC> | = An Escape character ('1B' hex) |
| Device | = The Zektor device being controlled. |
| Controller | = A PC or other system, used to control the Zektor device. |
| Parameter | = A decimal value (0-9) that may, in some cases, be prefixed with '+' or '-'. |

A K.I.S.S.™ command in its simplest form is a command following by a <CR> for instance:

V<CR>

This will return the version number of a Zektor device.

A command can have a variable number of parameters with optional whitespace(s) following the command, for instance:

P1<CR>

or

P 1<CR>

will turn power on. The spaces between the 'P' and '1' are optional. Since commands consist of alpha characters only, there can never be a 'P1' command and 'P1' will always be interpreted as 'P 1'.

When a command has more than one parameter, the parameters are separated by either whitespace(s) or a comma, or both whitespace(s) and a comma, for instance:

LI 3 3 80<CR>

or

LI 3,3 , 80<CR>

will set the front panel LEDs to auto-dim, with a low intensity setting of 3, and a high intensity setting of 80. Once again the space between the command and 1st parameter is optional. Space(s) may also appear before and after the comma.

The comma is optional between parameters except when it is necessary to indicate a default parameter, for instance:

LI , ,13<CR>

would set just high intensity level of the front panel LEDs without affecting the lower level, or the mode. The commas are used to indicate the 1st and 2nd parameters are not supplied and the default values should be used (in this case the default values are the current values, leaving the values unchanged). The space before the 1st comma is optional.

Most commands can be queried for their current settings by substituting the '?' for the parameter list, or by not supplying any parameters at all. For instance to request the current LED Intensity settings:

LI ?<CR>

or

LI<CR>

This would cause the device to issue a LED Intensity Response, (the Response String format is described in the paragraphs entitled: "The Response String"). The whitespace before the '?' is optional.

Using Bitmapped Parameters

Some commands accept “Bitmapped” parameters. These are decimal values that represent a series of flags, or bits, that control, enable and/or disable different device operations.

Binary arithmetic is used to represent bitmapped parameters, it is assumed the reader has some familiarity with binary arithmetic.

An example of a command that uses a bitmapped parameter is the “**XS settings<CR>**” command, which is defined as:

XS settings<CR>

Where ‘*settings*’ is a bitmapped parameter:

| Decimal Value | +2048 | +1024 | +512 | +256 | +128 | +64 | +32 | +16 | +8 | +4 | +2 | +1 |
|---------------|-------|-------|------|------|------|-----|-----|-----|-----|-----|-----|-----|
| Bit Position | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | AMU | VMU | 12V | AON | IRJ | IRS | IRE | BTE | SET | CRC | CSE | ASY |

Factory Settings

0 0 0 1 1 1 1 0

ASY - 0=Master / Slave mode.

1=Asynchronous Mode.

CSE - 0=Disable the sending of CS's or CRC-8's.

1=Append either Checksums or CRC-8 to responses.

CRC- 0=Append Checksums to responses if 'CSE' set.

1=Append CRC-8's, to responses if 'CSE' is set.

SET - 0=Disable the setup option.

1=Enable the setup option.

BTE - 0=Use 'BE' settings for front panels buttons.

1=Enable all front panel buttons, overrides 'BE' settings.

IRE - 0=User 'IRE' settings for IR control.

1=Enable all IR commands, overrides 'IRE' settings.

IRS - 0=Disable IR sensor.

1=Enable IR sensor.

IRJ - 0=Disable IR jack.

1=Enable IR jack.

AON- 0=Disable auto-on with any button press.

1=Enable auto-power on with button press.

12V - 0=Disable 12V control.

1=Enable 12V control.

VMU- 0=Disable video muting when a zone is muted.

1=Enable video muting when a zone is muted.

AMU- 0=Disable audio muting when a zone is muted.

1=Enable audio muting when a zone is muted.

The ‘Decimal Value’ in the table’s header, refers to the values added together to create the decimal parameter used by the command. For instances if the bits ‘ASY’ and ‘IRS’ where to be set to 1, and the rest of the bits set to zero, the parameter value would be calculated as: 1+64, making the parameter value: 65.

The command to directly set those two bits, and reset all the others would be:

XS 65<CR>

Individual bits of a bitmapped parameter can be set or reset without affecting the other bits, by prefixing the bitmapped parameter with a ‘+’ to set individual bits, or a ‘-’ to reset individual bits.

For instance in the above example the bitmapped value has been set to ‘65’. If we would now like to enable the IR jack, by setting the ‘IRJ’ bit, the following command can be issued:

XS +128<CR>

This will set the ‘IRJ’ bit, and have no effect on the others, and the new “XS” value would be: 193

If we’d like to now disable the IR jack and the IR sensor, by clearing the ‘IRJ’ and ‘IRS’ bits, we’d use the value “128+64, or 136 and issue the command:

XS -192<CR>

leaving the new “XS” value to be: 1.

Appending a Checksums or CRC-8 Checkcode to a Command

A checksum or CRC-8 checkcode may be appended to any command, and if given, will be calculated by the device and compared with the given value. If a mismatch occurs an error will be returned and the command will not be executed. This can be used to help assure reliable operation in noisy environments. Checksums are more commonly used in serial protocols, however CRC-8 checkcodes offer a more secure means of insuring error free communications.

Appending a Checksum or CRC-8 Checkcode to a Command (Continued)

A checksum or CRC-8 checkcode is appended to the command by adding a semicolon (;) for a checksum, or a colon (:) for a CRC-8 checkcode.

An example of appending a checksum to a command:

```
LI 2,13;178<CR>
```

the ';' indicates a checksum follows, the '178' is the checksum of the command string up to and including the ';' character.

In a similar fashion a CRC-8 checkcode can be appended to a command:

```
LI 2,13:213<CR>
```

The ':' indicates that a CRC-8 checkcode follows, the '213' is the calculated CRC-8 checkcode up to and including the ':' character.

Optional spaces are allowed before the ';' and ':' characters but NOT after them. The checksum must immediately follow the ';' character, and a CRC-8 checkcode must immediately follow the ':' character, anything else, including whitespace, will cause a syntax error to be returned. Similarly the <CR> must immediately follow the checksum or checkcode parameter or a syntax error will be returned.

See: ["Calculating Checksums and CRC-8 Checkcodes" on page 30](#) for more information on calculating checksums and CRC-8 checkcodes, it includes source code examples of calculating both Checksums and CRC-8's as used by K.I.S.S.™.

Clearing the Command Buffer

All commands are buffered and nothing is executed until the <CR> character is received. To assure that there are no extraneous characters in the command buffer, before a command string is sent, the <ESC> character can be issued to clear the buffer and reset any checksum or CRC-8 checkcode calculations.

This is useful when communications with the Zektor device is being initialized and the state of the device is unknown. An <ESC> will clear the command buffer and reset all checksums and CRC-8 checkcodes.

For example:

```
dsLG%df<ESC>V;145<CR>
```

will return the Version Query Response string. The "dsLG%df" represents noise that could have been in the buffer before the command string was issued. The <ESC> clears the buffer allowing the "V;145<CR>" to be processed error free.

It is legitimate (though seldom necessary) to prefix all commands with the <ESC> character to assure the buffer is always empty before the command string is received, which may be helpful in some very noisy environments.

Response Strings

A response will always be returned whenever a <CR> is received. There are no conditions where a "time-out" is a valid response to any query.

There are only three valid responses in the K.I.S.S.™ protocol, anything else should be considered a communication error, including a time-out while waiting for a response.

Each response is prefixed by a unique character. Determining which of the three responses is received can be done simply, by examining only the first character of any response string.

The three possible prefix characters and their associated responses are

- + The Acknowledgement Response
- ! The Error Response
- = The Query Response

The response to a command string will always be an Acknowledgement or an Error Response.

Response Strings (Continued)

The Acknowledgement is always the string:

+<CR><LF>

and the Error Response is always the string:

!ERR<CR><LF>

By parsing only the prefix characters '+' and '!', a programmer can choose to ignore the error codes and simply look at the first characters of the response strings and use them as a pass / fail indicator when issuing a command.

All response strings always end with a <CR><LF>.

A Query Response string always starts with the '=' characters and is followed by a command string indicating the parameter(s) being returned. This is better explained in an example.

An example of querying a device for its light intensity settings:

| | |
|---------------------------------------|--------------------------------------|
| LI?<CR> | Sent: Light Intensity Query command |
| +<CR><LF> | Received: Acknowledgement of command |
| =LI 3,2,80<CR><LF> | Received: The 'LI' Query Response |

Note that a "+<CR><LF>" followed the command string. A command string is always followed by either an Acknowledgment (as in this case) or an Error Response. This consistency allows a driver to use a single routine to issue a command and check for an Acknowledgment or an Error Response String, whether or not the command queries for a response.

An example of an error response:

| | |
|-------------------------------|---|
| IL?<CR> | Sent: Command characters transposed (no such command) |
| !2<CR><LF> | Received: Error Response indicating unknown command |

In this case the Error Response string "!2<CR><LF>" was issued instead of the acknowledgment string since the command was not recognized.

Response String Checksums and CRC-8 Checkcodes

Response strings can be programmed to have checksums or CRC-8 checkcodes appended to them, the syntax is identical to the Command Structure's checksum and CRC-8 handling.

Checksum and CRC-8 are turned on and off by issuing the proper command. On most devices this is the "Control Settings" command.

Only the Error Response and The Query Response strings will have checksum and CRC-8 checkcodes appended to them. The Acknowledgment Response will always consist of "+<CR><LF>". Anything else must be assumed to be a communication error.

An example with checksumming enabled, while querying for LED intensities is:

| | |
|--|----------|
| LI?<CR> | Sent |
| +<CR><LF> | Received |
| =LI 3,2,80;21<CR><LF> | Received |

An example with CRC-8 responses enabled is:

```
LI?:194<CR>
+<CR><LF>
=LI 3,2,80:207<CR><LF>
```

Response String Checksums and CRC-8 Checkcodes (Continued)

Notice that in the first example a checksum was not appended to the “LI?” command. When issuing a command the checksum and CRC-8 codes are sent on a command by command basis. Anytime a checksum or a CRC-8 code is appended to a command it will be checked and validated by the device, regardless of the “Response Checksum / CRC-8” settings.

When Response Checksums, or Response CRC-8 Checkcodes are enabled, Error Response strings will also have checksum or CRC-8 codes appended to them.

Master / Slave and Asynchronous Modes of Operation

The K.I.S.S.™ protocol can be used in a Master / Slave mode, where responses are only sent when requested, or in an Asynchronous mode, where responses are sent whenever the operational state of the device changes, such as a front panel button being pressed.

The Master / Slave Mode of Operation

In the Master / Slave mode, the controller requests information from the device at polled intervals. The control program assumes the role of the master, and the device is operated in the slave mode. No information will be sent from the device without first receiving a request from the controller.

For controllers that cannot handle having information being sent to them in the background, or at unspecified times, the Master / Slave setting is ideal, since all state changes will be logged but not sent until requested by the controller.

To allow for more efficient Master / Slave operations, there is a Query Status command available to the controller that return the status state of the device as a bitmap of flags indicating which states have changed and need querying. This allows the controller to poll, using a single command, and then based on those flag settings, issue only the commands needed to read the new state(s) of the device.

The Master / Slave mode also allows for a predictable communications flow. A communication sequence is always started by the controller by issuing a command. The response will always be either the Acknowledgement Response, or an Error Response, followed by (if a query command was issued) the Query Response. For instance:

| | |
|--------------------|---|
| LI ?<CR> | Sent: Controller issues a query command |
| +<CR><LF> | Received: Acknowledgment (or possible Error) Response |
| =LI 3,2,80<CR><LF> | Received: Query Response |

In the Master / Slave mode, the Acknowledgement or Error Response will always be the next response string after a command is issued, and a Query Response will always follow an Acknowledgement Response.

The Asynchronous Mode of Operation

In the Asynchronous mode the device sends a Query Response string anytime there is a state change. For instance when an input is changed by the user by pressing a button on the front panel, or using a remote control, or by the IR jack, or even if a serial command has been issued, a Query Response string indicating an input change will be sent to the controller.

The advantage if this mode is the controller can be kept constantly in sync with the device without having to send periodic polling commands. This disadvantages are: The controller must be able to receive the Query Response strings in the background at unspecified times, and the communications flow is slightly more complicated.

The Asynchronous Mode of Operation (Continued)

When issuing commands in the Asynchronous mode, the controller must be aware of any unsolicited Query Responses that may be interjected into the communications flow.

For example:

| | |
|---|--|
| <code>LI?<CR></code> | Sent: Controller issues query command |
| <code>=SZ 1,3</code> | Received: (unsolicited) Zone 1 remapped to Input 3 |
| <code>+<CR><LF></code> | Received: Acknowledgment Response |
| <code>=P 0</code> | Received: (unsolicited) Power turned off |
| <code>=LI 3,2,13<CR><LF></code> | Received: Query Response for LED intensities |

This represents a worst case scenario where unsolicited responses appear throughout the communication sequence.

First the “LI?<CR>” command was issued by the controller.

While looking for an Acknowledgement or Error Response string, an unsolicited Query Response is received, indicating the user has remapped Zone 1 to Input 3, while the “LI?<CR>” command was being issued by the controller.

Next the Acknowledgment Response of the “LI?<CR>” command is received.

Next an unsolicited Query Response is received indicating the power has been turned off.

Finally the Query Response indicating the LED intensities is received.

The K.I.S.S.™ command structure was designed to make the above scenario easy to deal with. Since all Query Responses start with the ‘=’ character, they can be handled asynchronously, as they are received. One approach would be to write a “Get Response” routine that handles all Query Response internally (by looking for the ‘=’ character), and only passing through non-query responses.

By using such a routine the above scenario becomes:

| | |
|------------------------------------|---|
| <code>LI?<CR></code> | Sent: Controller issues command |
| <code>+<CR><LF></code> | Received: Acknowledgement (or Error) Response |

The unsolicited Query Responses were handled internally by the new “Get Response” routine, and filtered from the communication flow, and only the Acknowledgement (or possible Error) Responses were allowed to pass. When the “=LI 3,2,13<CR><LF>” response is eventually received, it will be handled like any other unsolicited response.

Using K.I.S.S.™ in the Asynchronous mode is nearly as easy as using it in the Master / Slave mode, allowing for the creation of simple to write, but highly efficient device drivers.

Calculating Checksums and CRC-8 Checkcodes

The use of a checksums or CRC-8 checkcodes can increase the reliability of communications between the controller and any Zektor device.

A checksum is calculated by using an unsigned byte as an accumulator, and adding together all the ASCII characters of a command string, up to and including the ‘;’ character, while ignoring any overflow, and appending it as a decimal parameter to the end of the command.

A CRC-8 checkcode is calculated in a very similar way, but a CRC-8 algorithm is used instead of a simply adding together the ASCII characters. The CRC-8 byte is initialized to ‘FF’ hex, and the resultant value is sent inverted (one’s compliment).

The CRC polynomial used is: $x^8 + x^6 + x^3 + x^2 + 1$.

Calculating Checksums and CRC-8 Checkcodes (Continued)

This polynomial was determined through exhaustive tests (all 8 bit polynomials tested), to be the best CRC-8 polynomial for arbitrarily length bit streams.

See the paper entitled: "Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks" by Philip Koopman & Tridib Chakravarty:

<http://www.ece.cmu.edu/~koopman/roses/dsn04/koopman04_crc_poly_embedded.pdf>

Another good source of CRC information is the CRC entry on Wikipedia:

<http://en.wikipedia.org/wiki/Cyclic_redundancy_check>

Differences between a Checksum and a CRC-8 Checkcode

A CRC is capable of finding more, and different types of errors, than a checksum can. A good description of its capability is described in the above referenced articles, but a simple example shows some of the differences.

Here's an example of the intended command string:

LI 3,2,80

Here's some examples of the original and some badly formed strings, of the above example, and their associated checksums:

LI 2,3,80;21

LI 3,80,2;21

IL 3,2,80;21

KJ 80,2,3;21

Notice that every checksum is the same. Checksums cannot detect data being out of order. Checksums cannot detect errors where two bits, in the same position in two different bytes, are flipped. Checksums are not a very robust way to check for communication errors.

For comparison, here are the same examples and their associated CRC-8 checkcodes:

LI 2,3,80;84

LI 3,80,2;100

IL 3,2,80;127

KJ 80,2,3;9

The CRC-8 checkcode easily catches these errors.

Source Code Example of Calculating a Checksum

The following is a simple “C” program that calculates the checksum of the string “TestString” and then prints the initial string with the calculated checksum appended to it.

If you are using Acrobat reader to read this, a text version of the this sample is also available by double clicking the paper clip to the left of this paragraph.

Or you can download it at: <http://www.zektor.com/downloads/ckstst.c>

```
#include "stdio.h"

int main( void)
{
    char          TestString[] = "LI 3,2,80";
    unsigned char cksum;
    int           index;
    char          token = ',';

    cksum = 0;      // initialize checksum

    // Checksum all of 'TestString[]'

    index = 0;

    while (TestString[index] != '\0')
        cksum += TestString[index++];

    // Add the checksum token character ',' to checksum

    cksum += token;

    // Print the results

    printf( "%s%c%u", TestString, token, (unsigned char)cksum);
    return (0);
}
```

Source Code Example of Calculating a CRC-8 Checkcode

The following is a simple "C" program that calculates the CRC-8 of the string "TestString" and then prints the initial string with the calculated CRC-8 checkcode appended to it.

If you are using Acrobat reader to read this, a text version of the this sample is also available by double clicking the paper clip to the left of this paragraph.

Or you can download it at: <http://www.zektor.com/downloads/crc8tst.c>

```
#include "stdio.h"

// Routine for updating the CRC-8 checkcode using a polynomial
// of: x^8 +x^6 +x^3 +x^2 + 1.
//
// To create the CRC8_POLY mask, we start by ignoring the highest
// bit (x^8) since it is assumed to always be 1 and lies outside
// our byte boundary, and doesn't affect our results.
//
// The rest of the bits of the polynomial are reversed from the
// polynomial's order. This allows us to read in each bit starting
// with bit 0 of each byte, instead of bit 7. This is done because
// the UART sends its LSB first and by doing the same we are able to
// preserve the CRC's burst error detection characteristics.
//
// So:
//   x^8 +x^6 +x^3 +x^2 + 1 = 101001101 = 14D hex
//   Ignore X^8:             01001101 = 4D hex
//   Reverse bit order:      10110010 = B2 hex

#define CRC8_POLY    0xB2           // polynomial mask
#define CRC8_INIT    0xFF          // initial value

void crcByte( unsigned char *crc8, char cc)
{
    unsigned char  lcrc8;           // local copy of CRC-8 value
    int            bitcount;

    lcrc8 = *crc8;                 // get local copy of CRC-8

    // update CRC-8 with 8 new bits

    lcrc8 ^= cc;                   // test each bit against CRC-8

    for (bitcount = 0; bitcount < 8; bitcount++)
    {
        // if resultant bit was a 1, shift and xor in mask
        // else, just shift

        if (lcrc8 & 0x01)
            lcrc8 = ((lcrc8 >> 1) & 0x7F) ^ CRC8_POLY;

        else
            lcrc8 = (lcrc8 >> 1) & 0x7F;
    }
    *crc8 = lcrc8;                 // return new CRC8
}
```

Source Code Example of Calculating a CRC-8 Checkcode (Continued)

```
int main( void)
{
    char          TestString[] = "LI 3,2,80";
    unsigned char crc8;
    int           index;
    char         token = ':';

    crc8 = CRC8_INIT;           // initialize checkcode

    // CRC8 all of TestString[]

    index = 0;

    while (TestString[index] != 0)
        crcByte( &crc8, TestString[index++]);

    // Add the CRC-8 token character ':' to CRC-8

    crcByte( &crc8, token);

    // Finish with CRC8 by doing a one's compliment

    crc8 = ~crc8;

    // Print the results

    printf( "%s%c%u", TestString, token, (unsigned char)crc8);
    return (0);
}
```