Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

Propeller Backpack Video Overlay Object

General Description

The video overlay object for the Propeller Backpack provides the ability to overlay text on an NTSC video signal using a simple byte-oriented protocol. With the included overlay terminal program, BASIC Stamp (and other micro) users have the ability to control the overlay process via the Backpack's three-pin serial interface.

The Propeller Backpack (#28327) is a compact, multipurpose Propeller-based module that is optimized for audio and video applications. It includes the necessary hardware for fully self-contained use as an overlay controller. For expansion, it includes a socket to accommodate a daughterboard from Parallax's growing assortment of MoBo accessories.

Features

- Provides windowed character overlays.
- Multiple display lists provide rapid switching among various "screens" of data.
- User-selectable black/gray/white levels for character background and foreground, per window.
- User-selectable window transparencies.
- Automatic smooth scrolling option.
- Automatic word wrap option.
- NTSC video in; NTSC video out.

Applications

- Overlay GPS coordinates on remote sensing (e.g. underwater) videos.
- Record date and time on videos for documentation.
- Overlay instrument panel data on remote navigation screens.

Installation

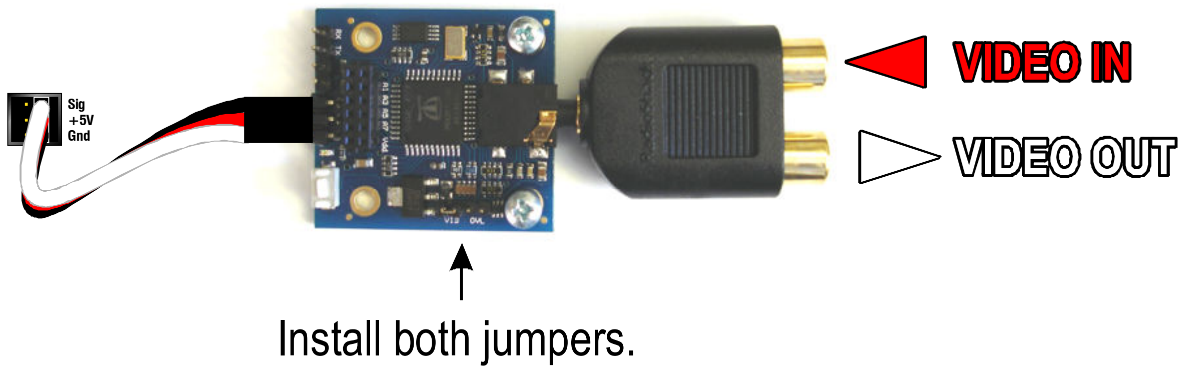
For this program, you will need:

- A source of NTSC video, such as a video camera.
- Propeller Backpack.
- Video monitor or recorder.
- BASIC Stamp (or other micro capable of 3.3 – 5V open-drain 9600-baud serial I/O) *or* a 5V power source and a Parallax Prop Plug.
- Adapter and cables to connect everything together.

Quick Start

BASIC Stamp Users

Before connecting any equipment together, install the included file, **prop_backpack_overlay_terminal.binary** into the Backpack's EEPROM. (Instructions for this are provided in the Propeller Backpack documentation.) Then, connect your equipment as follows:



In the photo above, a Radio Shack 1/8" stereo to 2xRCA adapter (#274-883) was used to provide separate RCA connectors for incoming and outgoing video. For this part, the red jack handles the incoming video; white, outgoing. Both the **VID** and **OVL** jumpers need to be installed.

The PBASIC program included with this documentation (**PropBP_overlay_demo.bs2**) demonstrates the use of the overlay terminal firmware. Once everything is hooked up, as shown above, you can run this program from any BASIC Stamp 2 family member. Make sure that the constant **io** is set to the actual BASIC Stamp pin number that you're using.

Prop Plug Users

Connect your equipment as shown above. You don't need a BASIC Stamp, but you do need a 5V power supply. Install the Prop Plug onto the 4-pin header, and upload the **PropBP_overlay_demo.binary** file included with this document. You will see the same demo as the one presented above. The source code for this demo is also included.

General Concepts

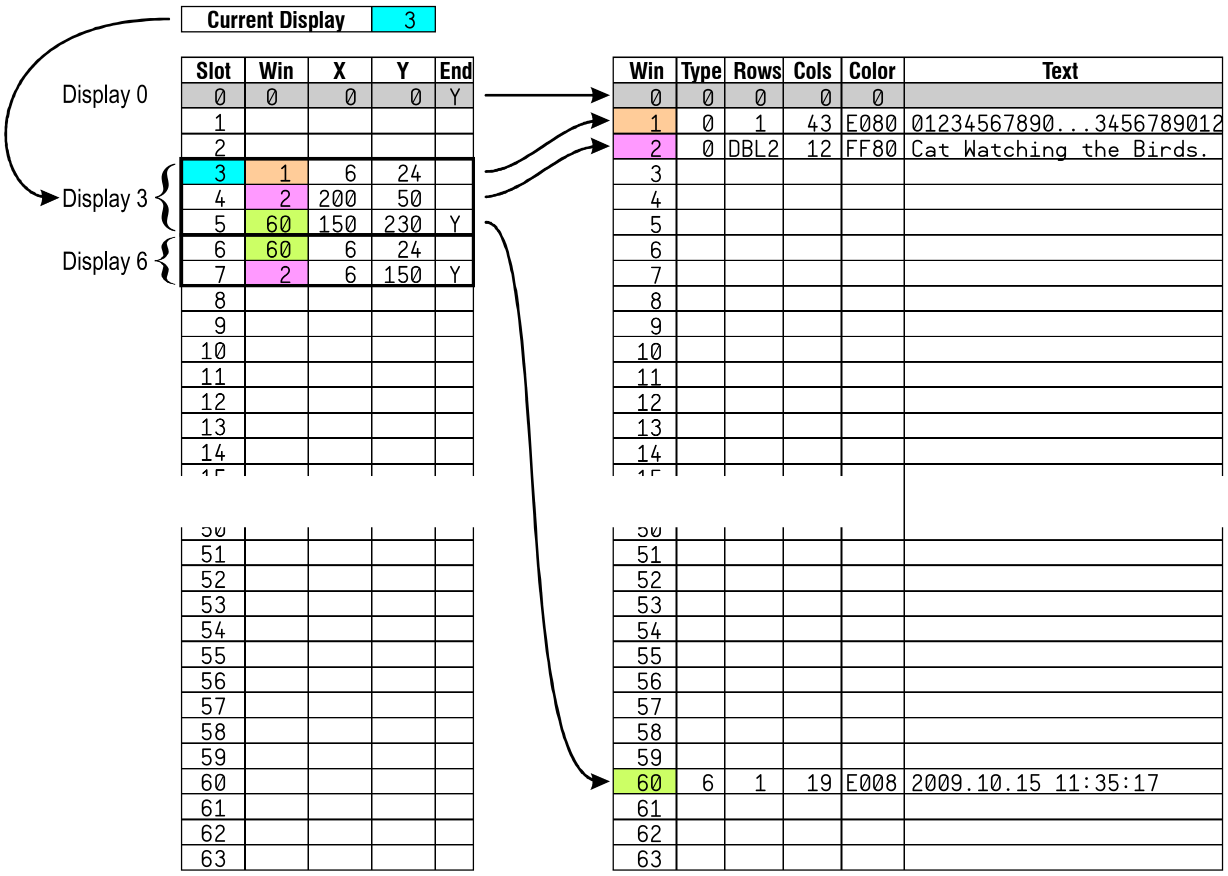
Before writing any programs that use the overlay object, there are some general concepts that apply to both PBASIC programs and Spin programs, and which pertain to how the overlay data are organized. These are outlined in the sections that follow.

Displays, Slots, and Windows

The top-level overlay unit is called a **display**. A display includes all the text that can be displayed on top of the incoming video. The overlay terminal program supports up to 64 displays, numbered from 0 to 63, from which one can be selected at any given time to display on the TV screen (or record to a VCR). Display **0** is predefined to be empty, so any time you want to remove all overlays from the incoming video, you can just show display **0**.

Text is not written directly to any display, however. It's written to windows. A **window** is a rectangular area within a display, defined by a height, width, character size, and "color" scheme. Since overlain text can only have various shades of gray ranging from black to white, "color" here refers to the foreground and background gray levels, as well as the degree of transparency against the incoming video background. A window's size and character size, once defined, cannot be altered. Its color, however, can be changed at will. There are 64 available windows, numbered from 0 to 63. Window **0** is predefined to be empty.

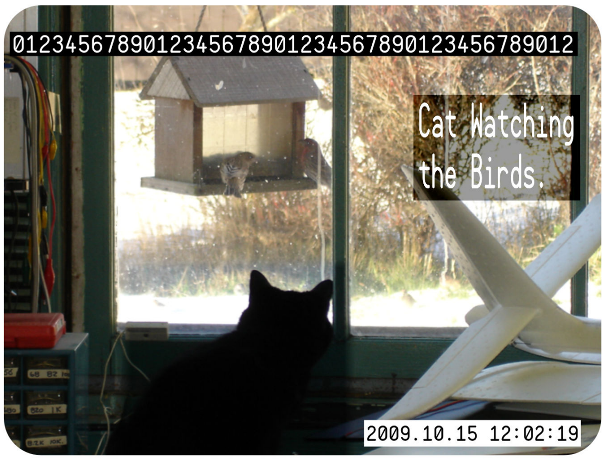
A display can include multiple windows. Windows are stacked vertically (with one or more blank lines of video between them). Therefore two windows cannot reside side-by-side or overlap in any way. Each window in a display is defined in a **slot**. There are 64 slots, and each slot contains information about which window is displayed, where it's being shown in the display, and whether or not it's currently visible. A display, then, is just a set of contiguous slots, the first slot pointing to the top window and successive slots pointing to windows further down. The last slot in a display is tagged, so the overlay firmware knows when the end of a display has been reached. The following diagram illustrates the relationships among displays, slots, and windows.



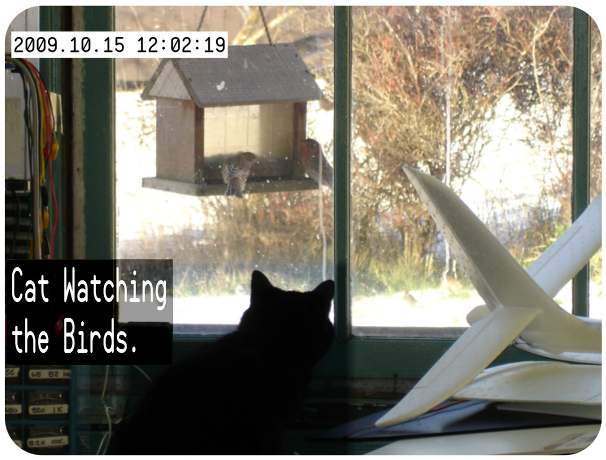
Slot	Win	X	Y	End
0	0	0	0	Y
1				
2				
3	1	6	24	
4	2	200	50	
5	60	150	230	Y
6	60	6	24	
7	2	6	150	Y
8				
9				
10				
11				
12				
13				
14				
51				
52				
53				
54				
55				
56				
57				
58				
59				
60				
61				
62				
63				

Win	Type	Rows	Cols	Color	Text
0	0	0	0	0	
1	0	1	43	E080	01234567890...3456789012
2	0	DBL2	12	FF80	Cat Watching the Birds.
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
51					
52					
53					
54					
55					
56					
57					
58					
59					
60	6	1	19	E008	2009.10.15 11:35:17
61					
62					
63					

Since display **3** is selected to show, the screen will look something like the illustration on the left, below. With display **6** selected, it will look like the one on the right.



Display 3



Display 6

Window Properties

A window possesses several properties that affect how it's displayed on the screen. These will be covered one at a time in this section.

Size

A window is a rectangular area whose size is given in **columns** (horizontal characters) by **rows** (vertical characters). Also affecting its physical size is the size of the characters within the window. These may be **single-wide** or **double-wide**, **single-high** or **double-high**. The *minimum* size for any window is two columns by one row. The *maximum width* depends on the character width. For single-wide characters it's 44 columns; for double-wide characters, 21 columns. The *maximum height* can be as much as 127 rows, regardless of character height, but no more than the top 13 or 14 rows will ever show up on the screen, so there's no reason to specify a greater height than that. Once a window's size parameters are defined, they cannot be changed.

Color

A window's "color" is specified by four different parameters, each of which can range from 0 to 15. These are mask, transparency, foreground, and background.

The **mask** is a four-bit number that defines which Propeller output pins are active when the window is displayed. The bits and their meanings are shown below:

Bit 3	Bit2	Bit1	Bit 0
Intensity (0 to 7)			Pass-through (0 or 1)

The intensity bits define the brightness range of the characters' foreground and background. Typically, these bits are all set to one, yielding a full range of brightness from 0 to 7. The pass-through bit specifies whether the foreground and background values can include transparency or not. If the bit is set to one, character transparency will be allowed; if it's cleared to zero, transparency is wholly dependent on the overall transparency setting.

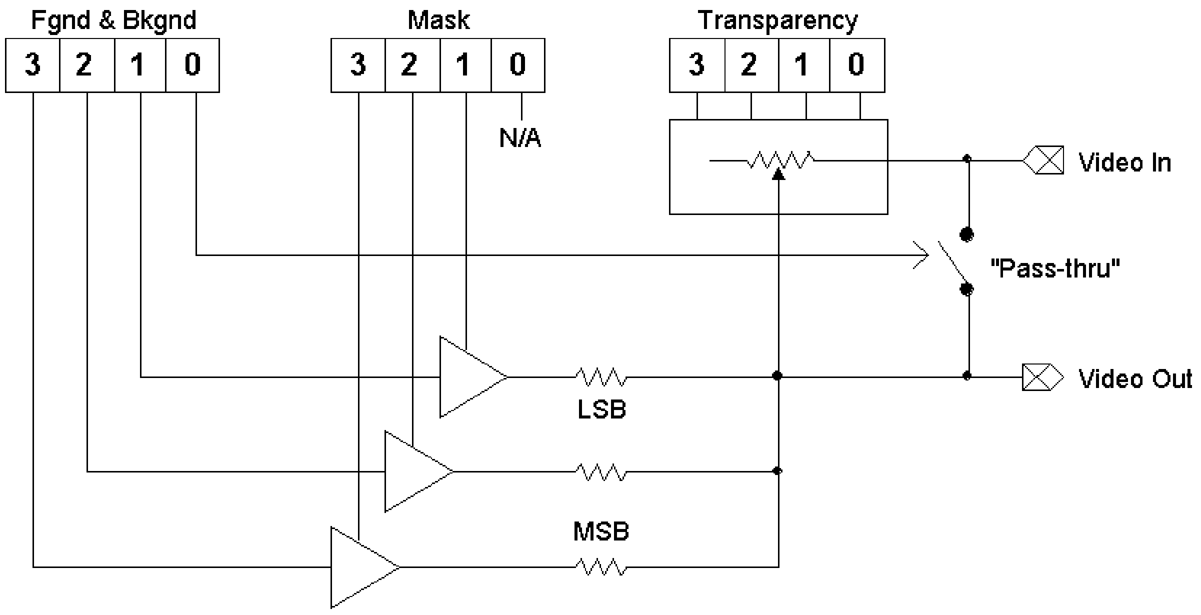
The **transparency** setting determines how much of the incoming video shows up behind the window. It can range from zero (none) to fifteen (a lot). If you want to block completely the incoming video from showing up behind the window, use a value of zero. Otherwise, successively higher values will let more of the incoming video through.

Note: Some super-cheap video cameras (especially certain "board cameras") and certain other video sources, including Parallax's Propeller Demo Board, output a higher than 2V peak-to-peak video signal when not terminated by a 75-ohm load. When the incoming video signal is blocked by a completely opaque window, it is effectively disconnected from its load at the video monitor. This can raise havoc with sync detection in the Propeller Backpack if the signal levels surpass 2V peak-to-peak. When using such video sources, the transparency should be set to eight or higher to maintain a load on the incoming video at all times, thus preventing loss of sync.

The **foreground** parameter determines the brightness of the characters themselves. It has the same format as the mask: an intensity number (bits 3 to 1), followed by a pass-through indicator (bit 0). Intensity ranges from black (0) to extra white (15). When the pass-through bit is high, the incoming video is allowed to pass through to the monitor unimpeded, albeit with added or diminished brightness, depending on the intensity number. When this bit is a zero, the amount of incoming video that passes through will be determined solely by the transparency setting, as described above.

The **background** parameter works the same as the foreground value but operates, instead, on the window's background.

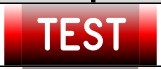
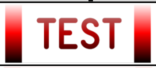
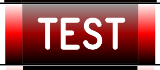
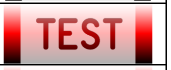
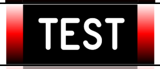
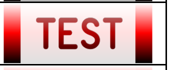
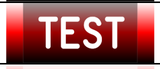
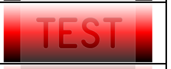
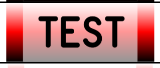
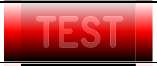
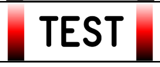
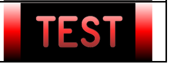
Here's a simplified block diagram of the video overlay circuitry that's active during a window display:



When displaying an active window, the three MSBs of the foreground and background values that are selected by the mask control the brightness of the characters and background that are overlain on the incoming video. Bit 0, whether selected by the mask or not, simply gates the pass-thru switch "on" (if high) and "off" (if low). When the pass-thru switch is open, the amount of input signal that gets to the output is governed by the transparency value.

Outside of an active window, the pass-thru switch remains closed, and the mask is set to zeroes, effectively disabling the three-resistor DAC and allowing the "video in" to pass to "video out" unimpeded and unmodified.

Now, let's see what different combinations of these four parameters can produce. In the following table, values for the four parameters are shown in hexadecimal, along with a test image overlain on a gradient background.

Mask	Trans.	Fgnd.	Bkgnd.	Sample	Mask	Trans.	Fgnd.	Bkgnd.	Sample
$F	$F	$8	$0		$F	$0	$1	$8	
$F	$8	$8	$0		$F	$0	$1	$9	
$F	$0	$8	$0		$F	$0	$1	$F	
$F	$F	$0	$8		$3	$0	$1	$3	
$F	$8	$0	$8		$3	$0	$3	$1	
$F	$0	$0	$8		$1	$0	$1	$0	

Scrolling

Windows one row high can be scrolled left and right; windows more than one row high can be scrolled up and down. Scrolling is in increments of 1/16 of a character width or 1/16 of a character height. Each window keeps track of its own scroll position.

Windows can also be set for automatic smooth scrolling. Each window has a **scroll rate** parameter which, when set to a non-zero value, enables smooth scrolling. The value itself controls the speed at which new characters enter a one-row window from the right (i.e. a marquee) or the speed at which a full line on the bottom row of a multi-row window moves up to make way for the next line. The scroll rate is given in milliseconds per scroll increment.

Word Wrapping

To ease the display of text, each window can do automatic word-wrapping when its word-wrap bit is set. When word wrapping is selected, contiguous non-blank characters (i.e. words) that extend past the end of a line are moved in their entirety to the next line in order to remain unbroken. Here is some sample text with word-wrapping turned off and with it turned on:

Without Word-wrapping	With Word-wrapping
Four score and s	Four score and
even years ago,	seven years ago,
our fathers brou	our fathers
ght forth on thi	brought forth on
s continent a ne	this continent a
w nation, concei	new nation,

Blinking

The foreground of any window can be set to blink on and off at a 1 Hz. rate. This is a feature that should be used sparingly and is most often employed to call attention to an emergent situation.

Window Type

Most windows are of **type 0**, i.e. ordinary text windows. Windows 60 through 63 can also be of types **1** through **8**. These are special types used to keep track of and display date and/or time information in the following formats:

Type	Name	Date/Time Format
1	HMS24	23:59:59
2	HMS12	11:59:59pm
3	YMD	2099-12-31
4	MDY	12/31/2099
5	DMY	31.12.2099
6	YMDHMS24	2099-12-31 23:59:59
7	MDYHMS12	12/31/2009 11:59:59pm
8	DMYHMS12	31.12.2009 11:59:59pm

Once initialized, time and date windows will keep time and update automatically, taking into account the lengths of months and leap years. The range of time that can be accommodated is from **2000-01-01 00:00:00** to **2099-12-31 23:59:59**, and all date/time windows will track the same time.

Display and Slot Properties

A display is just a sequence of slots, ordered from top to bottom. Therefore, the *relative* positions of the windows pointed to by the slots are fixed by the slot order. Each slot has its own properties, which are defined below.

Window Number

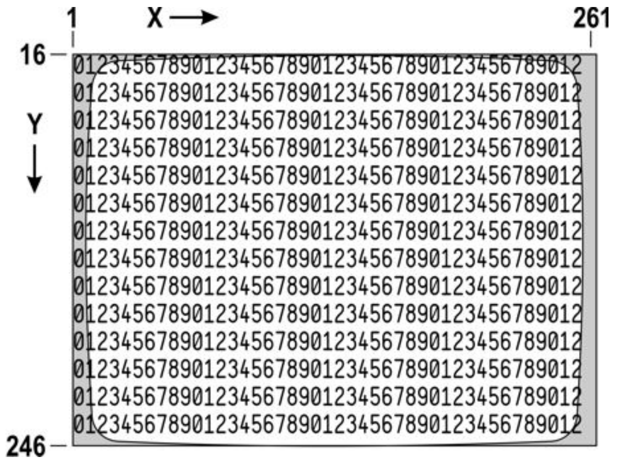
Each slot points to one window. The window number (0 – 63) determines which window is displayed at the slot's relative position in the display. If the window number is zero, nothing is displayed. The window number can be changed at any time. This enables instantaneous switching between different texts within a given display. It also enables double-buffering, wherein two windows contain the same data, but one is shown while the other is being updated, then swapped.

Window Visibility

The window in each slot can be turned "on" or "off". If it's "off", it will still occupy space on the screen, but it will be completely transparent, and no characters will be visible.

Window Position

The screen position of any window is specified by the X and Y coordinates of its upper left-hand corner. The screen's coordinate system is shown in the following diagram, along with a window of 43 columns by 14 lines, shown in its leftmost and uppermost displayable position (1, 16):



This illustrates the maximum extent of a displayable screen. Most real displays – especially those employing CRTs – are *overscanned*, which means that the displayable area extends beyond the edges of the physical screen (the gray area in the above illustration). In such systems, the maximum practical window size is 40 columns by 13 rows, beginning at location (15, 27). But each display is different, so experimentation is the only way to tell how big an area it can show.

Each individual character in this coordinate system are six units wide and sixteen units tall. The rightmost position that can be displayed is at X = 261, and the firmware will not allow a window to extend beyond this point. If it does, it will be moved back to align with this rightmost edge. Therefore, in order to keep control over a window's horizontal position, the X coordinate for the left edge should be no larger than 261 – 6 * width. The Y coordinate of the top edge can be anything from 1 to 254. This means that windows can be either too high or too low to display in their entirety, in which case they're simply cropped.

A special case exists when the X and/or Y coordinate of a window position is zero. This indicates to the system that you want the window centered on that axis, and the firmware will perform the necessary calculations to do so.

Because two windows cannot exist side-by-side or overlap in any way, there can be times when a displayed window is specified to begin above the bottom of the window preceding it. For example, if a four-line window began at Y = 30, and if the next window were slated to begin at Y = 75, there would be a potential conflict, since the first window would end at Y= 30 + 4 * 16 – 1 = 93. The software resolves this conflict by starting the second window at Y = 95, leaving one blank scan line (at Y = 94) between them. Therefore if you want to pack multiple windows vertically as close as possible, you can specify Y = 1 for the top edges of all but the first window and let the firmware do the vertical packing for you.

Last Slot

Every display has a "last slot", which has a bit set to indicate that there are no more slots to display. This bit is set automatically by the firmware as slots are appended to a display. There is no way for the user to modify it.

Overlay Terminal Program

The program that BASIC Stamp programmers, or anyone else doing overlays via the three-pin serial port, will use is **prop_backpack_overlay_terminal.spin**, which calls **prop_backpack_tv_overlay.spin**, along with **basic_io.spin**. The easiest way to get going is to use the **LoadPropBP.exe** program downloadable from the Propeller Backpack product page at parallax.com and install the binary version of the terminal program, **prop_backpack_overlay_terminal.bin**, included with this documentation. (You can also download the Spin source code from the Propeller Object Exchange, if you like.)

Once the terminal firmware has been installed, you're ready to start programming. Communication with the overlay terminal program takes place with serial I/O, transmitting and receiving at 9600 baud, using a **non-inverted**, **open-drain** protocol. The Backpack includes a pullup resistor to +5V, so you do not need to apply one externally. Because of the open-drain protocol, *the pin used to communicate with the Propeller Backpack should always be configured as an input, except when being driven low.*

The pin and baudmode settings for the BS2 on pin 15 at 9600 baud, for example, would be:

```
io    PIN 15
baud  CON 84 + 32768
```

When your PBASIC program first starts, it is a good idea to reset the Propeller Backpack before doing anything else. You do this by pulling the serial I/O pin low for 10ms and releasing it. Do not hold it low for more than a second, though, as this will cause the Backpack to enter "programming mode" and not start the terminal program. Here's a typical startup sequence:

```
LOW io            'Reset the Propeller Backpack
PAUSE 10
INPUT io

PAUSE 2000        'Wait for it to come out of reset.
```

You should always copy the constants and startup code from the BASIC Stamp demo program, **PropBP_overlay_demo.bs2**, to any program you intend you use with the overlay terminal firmware. Using the command constants defined there will make your programs more readable.

When the overlay program resets, there will be no defined displays, slots, or screens, except for the null screen **0** and the null window **0**. Your first job will be to define them before writing text to the screen. In the sections to follow, the commands necessary for using the terminal program are presented in a logical order. Each command is given with both its defined constant name and its hexadecimal value. Where the term “cursor” is used, it refers to the position where the next character will be written.

Overlay-specific Commands

PRESET	**$1C, disp, n**	**Select one of the built-in display presets.**

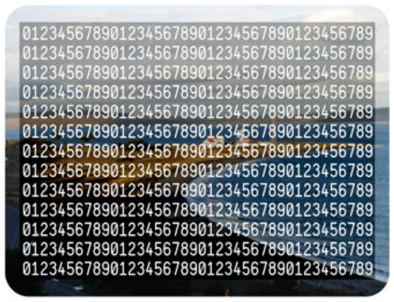
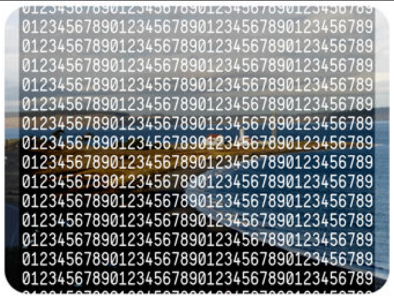
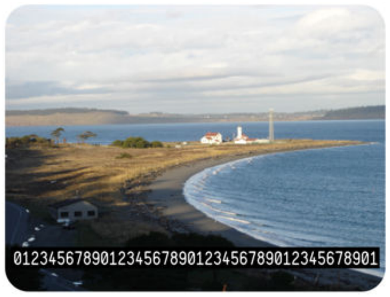
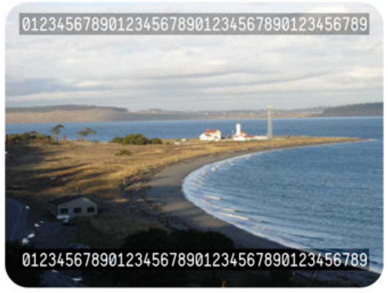
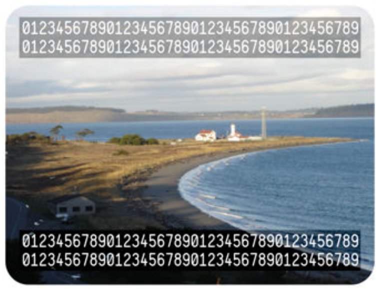
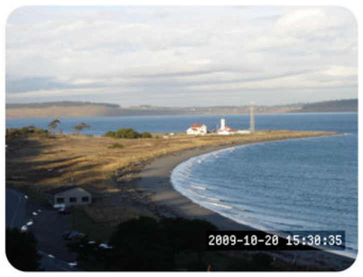
The **PRESET** command provides the quickest way to get started using the overlay. Using this command you can select from several preconfigured displays and simply start writing text. **PRESET** needs to be followed by two bytes, **disp** (a display number between **1** and **63**) and **n** (the number or name of the selected preset). In the list of constants, the presets are all given names, hinting at how they might be used.

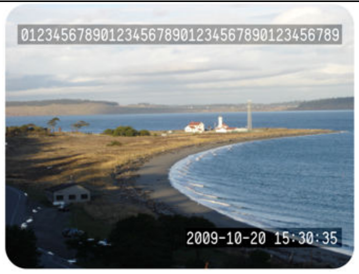
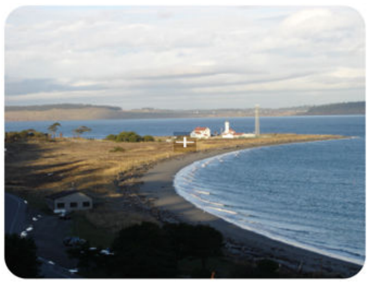
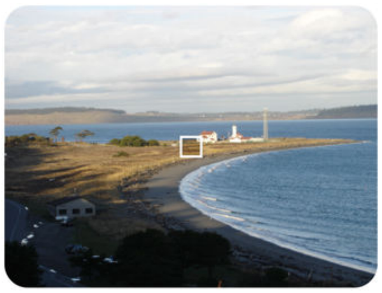
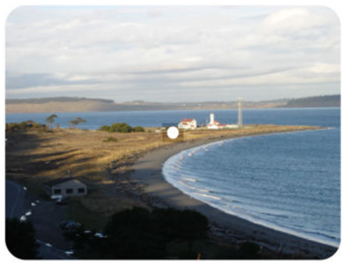
To get started, let’s select a preset that creates a scrolling marquee across the bottom of the screen as display **5** and displays some text there:

```
SEROUT io,bd, [PRESET, 5, MARQUEE]
SEROUT io,bd, [REP " "\43, "Testing the new marquee.", REP " "\43]
```

That’s all there is to it. When you run this program, you should see the text, "Testing the new marquee.", scrolling across the bottom of your screen in white text on a black stripe superimposed atop the incoming video.

The available presets are as shown in the table below. The window number is the same as the selected display number (**disp**), except where noted. In the case of multiple windows, the **USEWIN** command discussed later can be used to toggle between them. For date and time windows, see the **SETTIM** command.

 **BIGWIN**: One 40x13 window.	 **CREDITS**: One 40x17 window with smooth scrolling.	 **MARQUEE**: One opaque 42x1 window with smooth scrolling.
 **HILO**: Two 40x1 windows, top (**disp**) and bottom (**disp+1**).	 **HILO2**: Two 40x2 windows, top (**disp**) and bottom (**disp+1**).	 **DATETIME**: Date and time in lower right (**60**).

 0123456789012345678901234567890123456789 2009-10-20 15:30:35 **HIDATTIM**: 40x1 window at top (**disp**), date and time in lower right (**60**).	 **CROSS**: 2x1 cross-shaped cursor in the middle of the screen.	 **BOX**: 2x1 square cursor in the middle of the screen.
 **DOT**: 2x1 dot-shaped cursor in the middle of the screen.	The preceding three presets are cursors, which can be moved about by the **MOVWIN** command discussed later. Their initial position is (132,116), which centers them on the screen and which can be used as a reference for later repositioning. Such movable cursors provide a means to highlight points of interest in a presentation, for example, when the location can be coupled to a mouse or other positioning device. In order to cover the entire screen, cursors should be used alone, without other windows present to restrict their movement.	

DEFWIN	**$10, winno, cols, rows**	**Define window number winno (1 – 63), of dimension cols x rows.**

The **DEFWIN** command creates a window, which can be shown on one or more displays (i.e. plugged into one or more slots), and reserves memory for its text. Once created, a window can be neither resized nor destroyed. Its horizontal and vertical dimensions are given by the **cols** and **rows** arguments, respectively. For normal windows, there must be at least 2, and no more than 44 columns for single-width characters or 21 columns for double-width characters, and at least one row. Although the row count can extend to 127, no more than 14 or 15 rows will be visible, so there's seldom any point in going beyond that number. Upon creation, the newly defined window automatically becomes the "current window" for subsequent changes and text operations.

The **cols** and **rows** arguments can be modified to produce predefined date and/or time windows and further modified to produce double-wide and/or double-high characters. To produce a date/time window, set the **cols** value to zero, and use the **rows** value to choose the window type from among those discussed in the previous section. For double-wide characters, bitwise OR the value **DBL** to the **cols** argument. For double-high characters, bitwise OR the value **DBL** to the **rows** argument. In the following example, three windows are created.

```
SEROUT io,bd, [DEFWIN, 1, 20, 4]
SEROUT io,bd, [DEFWIN, 2, 12, DBL|1]
SEROUT io,bd, [DEFWIN, 60, DBL|0, HMS24]
```

Window **1** is twenty columns wide by four rows high; window **2**, twelve columns wide by one row high, with double-height characters; window **60**, a 24-hour time window with double-wide characters.

Windows are always created with blinking, word-wrapping, and smooth scrolling turned off and with **mask**, **trans**, **fgnd**, and **bkgnd** values of **$F**, **$A**, **$A**, and **$0**, respectively. These can be changed, however, using the **BLINK**, **WDWRAP**, **SMSCRL**, and **CHGCLR** commands discussed later.

APNDSP	**$18, disp, winno, x, y**	**Append window winno (0 – 63) to the end of display disp (1 – 63) at position (x, y).**

The **APNDSP** command appends a new slot to the end of the display beginning at slot **disp** and inserts a reference to window **winno** to it. The position of the window's upper left-hand corner is given by the coordinates **x** and **y**. If either **x** or **y** is zero, the window will be centered on that axis. If **winno** is zero, the null window, which is just a placeholder, will be inserted instead.

Windows inserted in this fashion are, by default, not visible unless the value **SHO** is bitwise ORed with the window number. Once a slot has been appended to the display with **APNDSP**, it cannot be removed; however the **winno** and **xy** values can be changed at any time by the **CHGWIN** and **MOVWIN** commands described later.

A display created with **APNDSP** is not automatically shown on the screen. For this to happen, you need to execute the **SHODSP** command, described below, which chooses the display to put on the screen.

The following example expands upon the previous one, adding the defined windows to display **3**:

```
SEROUT io,bd, [APNDSP, 3, SHO|1, 12, 24]
SEROUT io,bd, [APNDSP, 3, SHO|2, 140, 100]
SEROUT io,bd, [APNDSP, 3, SHO|60, 0, 200]
```

Notice that, in each case, the **SHO** modifier is ORed with each window number to make it visible on the screen. After the above code is run, window **1** will be put into slot **3** (the first slot in display **3**), window **2** into slot **4**, and window **3** into slot **5**. If another window were appended later, it would go into slot **6**. Slots are always appended to a display in consecutive order, and you have to keep track of which windows are in which slots, in case you want to change or move them later. You also have to be careful not to collide with slots that are already defined. For example, if there were already another display that began in slot **5**, the last **APNDSP** in the example above would fail.

SETTIM	**$1D, yr, mo, day, hr, min, sec**	**Set the current date and time to 20yr-mo-day hr:min:sec**

When a date/time window has been included in the display, it's necessary to initialize the time so the firmware can keep track of it. The **SETTIM** command does just that. The time can come from many sources, e.g. a clock/timer chip, GPS, or manual entry. In the following example, the time is set to 3:27:46 p.m. on 20 October 2009.

```
SEROUT io,bd, [SETTIM, 09, 10, 20, 15, 27, 46]
```

SHODSP	**$07, disp**	**Show display disp (0 – 63) on the screen.**

The **SHODSP** command selects a display to show on the screen. **disp** is the slot number of the first slot in the desired display. If **disp** is set to zero or to any other undefined slot, the overlay will be removed from the video.

In the following example, display **3** from the previous examples is chosen:

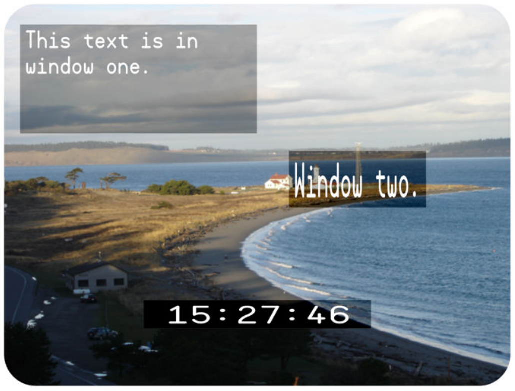
```
SEROUT io,bd, [SHODSP, 3]
```

USEWIN	**$11, winno**	**Window winno is made the "current window" for making changes and for subsequent text operations.**

The **USEWIN** command selects a window in which to perform subsequent changes and/or text operations. For example, to display text in window **2** from the above examples, and then in window **1**, you could use the following code:

```
SEROUT io,bd, [USEWIN, 2, "Window two."]
SEROUT io,bd, [USEWIN, 1, "This text is in", CR, "window one."]
```

After all the foregoing examples have been executed, this is what will result:



MOVWIN	**$19, slotno, x, y**	**Move the window in slot slotno to location (x,y).**

The upper left-hand corner of the window in the indicated slot is moved to the indicated **x** and **y** coordinates. The coordinates have the same range and meaning as those in the **APNDSP** command.

The following example shows how to move a box cursor around the screen. It uses the center (132, 116) as virtual location (0, 0), a runs the box around it in a clockwise circle. Although the coordinates are programmed in this example, they could well come from a positioning device, such as a mouse, instead.

```
x       VAR Byte
y       VAR Byte
angle   VAR Byte

SEROUT io,bd, [PRESET, 1, BOX]
DO
  IF (angle < 64 OR angle > 192) THEN
    x = COS(angle) >> 1 + 132
  ELSE
    x = 132 - (COS(angle - 128) >> 1)
  ENDIF
  IF (angle < 128) THEN
    y = SIN(angle) >> 1 + 116
  ELSE
    y = 116 - (SIN(-angle) >> 1)
  ENDIF
  SEROUT io,bd, [MOVWIN, 1, x MAX 254, y MAX 254]
  angle = angle + 1
LOOP
```

One thing to be careful of, as this example shows, is never to let either coordinate equal **255**. The overlay software treats **255** as zero to accommodate string operations in Spin, where a zero byte is the string terminator.

SHOWIN	**$1A, slot, yn**	**Show the window in slot, depending on yn.**

This command turns the window in slot number **slot** on or off, depending on the value of **yn** (“off” = 0, “on” = non-zero). When the window is “off”, it will still occupy the allotted space, but neither the foreground not background will be visible.

In the following program, the message, “Have a snack,” flashes briefly every ten seconds. (Such subliminal messages were used years ago as advertising, until they were made illegal.)

```
SEROUT io,bd, [DEFWIN, 1, DBL|13, DBL|1, "Have a snack."]
SEROUT io,bd, [APNDSP, 5, 1, 0, 0, SHODSP, 5]
DO
  SEROUT io,bd, [SHOWIN, 5, 1]
  PAUSE 30
  SEROUT io,bd, [SHOWIN, 5, 0]
  PAUSE 10000
LOOP
```

CHGWIN	**$1B, slot, winno**	**Change the window in slot to winno.**

The **CHGWIN** command changes which window a given slot points to. It also controls the visibility of the chosen window by ORing (or not) the value **SHO** to the window number.

Here’s a program that alternates between two messages by switching windows.

```
SEROUT io,bd, [DEFWIN, 1, DBL|16, DBL|1, " Enjoy a snack. "]
SEROUT io,bd, [DEFWIN, 2, DBL|16, DBL|1, "Visit the lobby."]
SEROUT io,bd, [APNDSP, 5, 1, 0, 0, SHODSP, 5]
DO
  SEROUT io,bd, [CHGWIN, 5, SHO|1]
  PAUSE 2000
  SEROUT io,bd, [CHGWIN, 5, SHO|2]
  PAUSE 2000
LOOP
```

NOTE: Because the windows are centered, they’re made the same size. This is because centered windows are assigned a computed position for their upper left-hand corners, and the fact that they are centered is forgotten. If a smaller window were exchanged for a larger one, or *vice-versa*, they would share the same corner position, but one of them would no longer be centered, due to the size difference. For single-line windows, if the difference between the windows’ character counts is odd, the shorter text can be scrolled left by **8** to recenter it. See the **SCROLL** command for details.

CPYWIN	**$17, winno**	**Copy the text from winno to the current window.**

This command is used to copy the contents of another window to the current window. Both windows must have the same column and row dimensions for the copy to take place. Almost all properties and text of the window being copied from are transferred, including scroll info, color, and cursor position, but excepting window type, blink state, character size, and word wrap flag. **CPYWIN** can be used to facilitate double-buffering, so that a sequence of gradual changes in a window's contents can update instantaneously on the screen.

In the following example, windows **1** and **2** serve as a double buffer. The new text is added to the window that is *not* being displayed. Then, after every three lines, *that* window gets displayed, the current window gets switched, and the displayed window copied to *it*. New text is then added to the newly-assigned current window, while the other one is being displayed, and so on.

```
i   VAR Word
win VAR Byte

SEROUT io,bd, [DEFWIN, 1, 5, 8]
SEROUT io,bd, [DEFWIN, 2, 5, 8]
SEROUT io,bd, [APNDSP, 5, SHO|2, 0, 0, SHODSP, 5, USEWIN, 1]
win = 1
DO
  PAUSE 500
  i = i + 1
  SEROUT io, bd, [DEC5 i]
  IF (i // 3 = 0) THEN
    SEROUT io, bd, [CHGWIN, 5, SHO|win]
    win = 3 - win
    SEROUT io, bd, [USEWIN, win, CPYWIN, 3 - win]
  ENDIF
LOOP
```

CHGCLR	**$12, mask, trans, fgnd, bkgnd**	**Change the "color" of the current window to the indicated values.**

When a new window is defined using the **DEFWIN** command, its "color" defaults to **mask**: $0F, **transparency**: $0A, **foreground**: $0A, **background**: $00; in other words, light letters on a darker, semitransparent background. You can change the color scheme for the *current window* by using the **CHGCLR** command.

In this example, the transparency of the background fades in and out:

```
i   VAR Byte

SEROUT io,bd, [DEFWIN, 1, 10, 4, "Watch the changing  trans-    parency."]
SEROUT io,bd, [APNDSP, 5, SHO|1, 0, 0, SHODSP, 5]
DO
  FOR i = 1 TO 31
    SEROUT io,bd, [CHGCLR, $0F, ABS(16 - i), $0A, $00]
    PAUSE 100
  NEXT
LOOP
```

SCROLL	**$13, value**	**Scroll the current window left or up by the designated value (0 – 15).**

This command will set the shift value for the contents of the current window by the designated amount. For single-row windows, the shift is applied to the horizontal position; for multi-row windows, to the vertical position. A shift value of zero means "unshifted". Larger values shift successively further up or to the left, by 1/16 of a character width or height until, at 15, the maximal shift is achieved.

In this example, two windows get scrolled: one, back-and-forth; the other, up-and-down:

```
i   VAR Byte

SEROUT io,bd, [DEFWIN, 1, 10, 4, CR, "Watch the changing  scroll."]
SEROUT io,bd, [DEFWIN, 2, 11, 1, " Here, too."]
SEROUT io,bd, [APNDSP, 5, SHO|1, 0, 75, APNDSP, 5, SHO|2, 0, 200, SHODSP, 5]
DO
  FOR i = 1 TO 31
    SEROUT io,bd, [USEWIN, 1, SCROLL, ABS(16 - i)]
    SEROUT io,bd, [USEWIN, 2, SCROLL, ABS(16 - i)]
    PAUSE 100
  NEXT
LOOP
```

You will seldom, if ever, need to use the **SCROLL** command, since the **SMSCRL** command, described next, handles scrolling for you, automatically.

SMSCRL	**$14, value**	**Set the smooth scrolling rate to value (0 – 254 ms).**

The **SMSCRL** command enables (**value > 0**) or disables (**value = 0**) smooth scrolling for the current window. When enabled, the delay for each increment of horizontal or vertical scrolling is set to the **value** parameter, measured in milliseconds. In single-row windows, scrolling takes place from right-to-left for each character added to the end of the row. In multiple-row windows, scrolling takes place from bottom to top for each row added after the last row. With smooth scrolling enabled, it is important that your program be able to pace itself so as not to overrun the overlay program's input buffers. This is done by using the **MARK** command, as shown in the following example, and described next.

```
i   VAR Word

SEROUT io,bd, [DEFWIN, 1, 10, 4, SMSCRL, 20]
SEROUT io,bd, [DEFWIN, 2, 40, 1, SMSCRL, 20]
SEROUT io,bd, [APNDSP, 5, SHO|1, 0, 75, APNDSP, 5, SHO|2, 0, 200, SHODSP, 5]
DO
  i = i + 1
  SEROUT io,bd, [USEWIN, 1, DEC i, CR]
  SEROUT io,bd, [USEWIN, 2, DEC i, " "]
  SEROUT io,bd, [MARK]
  SERIN  io,bd, [WAIT(MARK)]
LOOP
```

One thing you will notice from running this example is that only one window scrolls at a time. This is due to the fact that smooth scrolling occurs in the foreground as incoming characters are being processed. therefore, for the best effect, only one displayed window should be enabled for smooth scrolling at a time.

MARK	**$1E, value**	**Request an acknowledgement.**

Use the **MARK** command, as in the foregoing example, to ascertain that the overlay software has finished performing all commands sent so far. The software will wait 10ms after accepting the **MARK** from its input buffer to echo it back. This will provide an adequate turnaround time for the BASIC Stamp program to begin monitoring for the **MARK**'s return. Therefore, the very next statement after sending a **MARK** should be a **SERIN** to await its echo, just in case the overlay software has already caught up and echoes the **MARK** immediately.

WDWRAP	**$15, yn**	**Turn word-wrapping on or off for the current window.**

In the program example above for **CHGCLR**, you may have noticed the odd spacing applied to the text, “`Watch the changing  trans-     parency.`” This was done to ensure that words did not get broken between lines. With word-wrapping turned on (**yn > 0**), the extra spaces would not have been necessary, since the overlay software would have supplied them automatically. Here’s an example (from Dickens’ *A Tale of Two Cities*) that displays lengthier text. Watch it run as it adjusts the incoming words in real-time so as not to break them apart. Compare that to what happens if you remove the **WDWRAP** command.

```
DATA @0, "It was the best of times, it was the worst of times, it was"
DATA " the age of wisdom, it was the age of foolishness, it was the"
DATA " epoch of belief, it was the epoch of incredulity, it was the"
DATA " season of Light, it was the season of Darkness, it was the"
DATA " spring of hope, it was the winter of despair, we had everything"
DATA " before us, we were all going direct to Heaven, we were all"
DATA " going direct the other way -- in short, the period was so far"
DATA " like the present period, that some of its noisiest authorities"
DATA " insisted on its being received, for good or for evil, in the"
DATA " superlative degree of comparison only.", 0

char  VAR Byte
addr  VAR Word

SEROUT io,bd, [DEFWIN, 1, 15, 10, REP CR\10, SMSCRL, 50, WDWRAP, 1]
SEROUT io,bd, [APNDSP, 5, SHO|1, 0, 0, SHODSP, 5]
addr = 0
DO
  READ addr, char
  IF (char = 0) THEN END
  SEROUT io,bd, [char]
  addr = addr + 1
  IF (addr & 31 = 0) THEN
    SEROUT io,bd, [MARK]
    SERIN io,bd, [WAIT(MARK)]
  ENDIF
LOOP
```

BLINK	**$16, yn**	**Turn blinking on or off for the current window.**

The **BLINK** command turns blinking on (**yn > 0**) or off (**yn = 0**) for the current window. Here’s an example:

```
SEROUT io,bd, [DEFWIN, 1, 20, 3, WDWRAP, 1, BLINK, 1]
SEROUT io,bd, ["Blinking can be very annoying. Use it sparingly."]
SEROUT io,bd, [APNDSP, 5, SHO|1, 0, 0, SHODSP, 5]
```

ESC	**$1F, char**	**Send char as-is, without interpreting it as a command.**

The Propeller’s character set includes special characters mapped into the area ($00 - $1F) normally reserved for control codes and used by the overlay software for commands. To send one of these command characters to be displayed, you need to precede it with an escape character, **ESC**.

Here's an example that displays all the command characters.

```
i VAR Byte

SEROUT io,bd, [DEFWIN, 1, 8, 4]
SEROUT io,bd, [APNDSP, 5, SHO|1, 0, 0, SHODSP, 5]

FOR i = 0 TO 31
  SEROUT io,bd, [ESC, i]
NEXT
```

PBASIC Pre-defined Commands

The remaining commands are identical to those defined by PBASIC for its DEBUG screen and operate the same when using the overlay software. They are presented here for review without additional examples of their use.

CLS	**$00**	**Clear the current window.**
CLRWIN	**$FF**	**Alternative to CLS when zero bytes can't be sent.**

Clear the current window, converting all character cells to the window's background color, and home the cursor to the window's top left-hand corner.

HOME	**$01**	**Move cursor to the current window's home position.**

This command moves the cursor to the top left-hand corner of the current window but does not change the window's contents.

CRSRXY, x, y	**$02, x, y**	**Move cursor to position (x,y).**

The two bytes following this command define the new position (column, row) of the cursor within the current window, starting from column zero, row zero in the window's home position. The contents of the window are unchanged.

CRSRLF	**$03**	**Move cursor one column to the left.**
CRSRRT	**$04**	**Move cursor one column to the right.**
CRSRUP	**$05**	**Move cursor one row up.**
CRSRDN	**$06**	**Move cursor one row down.**

Move the cursor in the direction indicated. Motion saturates (stops) at the current window's left, right, top, and bottom edges.

BKSP	**$08**	**Backspace.**

Move cursor to the previous character position in the window and erase the character there.

TAB	**$09**	**Tab.**

Insert as many spaces as necessary to get to the next column position evenly divisible by eight in the current window.

LF	**$0A**	**Linefeed.**

Move cursor down one line in the current window. If cursor was in the window's bottom line, scroll window up by one line and clear the bottom line using the current background color.

CLREOL	**$0B**	**Clear to the end of the line.**

Clear the current line (row) in the current window from the current cursor position to the end of the line, using the current background color. The cursor position remains unchanged.

CLRDN	**$0C**	**Clear down.**

Clear the current window from the current cursor position to the end, using the current background color. The cursor position remains unchanged.

CR	**$0D**	**Carriage return.**

Clear to the end of the current line in the current window using the current background color, then perform a linefeed. Cursor is moved to the beginning of the new line.

CRSRX, x	**$0E, x**	**Move cursor to column x.**

Move the cursor to the column given by the next byte (**x**) within the current row in the current window. If **x** is greater than or equal to the width of the window, position the cursor in the last column.

CRSRY, y	**$0F, y**	**Move cursor to row y.**

Move the cursor to the row given by the next byte (**y**) within the current column in the current window. If **y** is greater than or equal to the height of the window, position the cursor in the last row.

Spin Programming

Spin programmers can access the overlay object directly, without relying on an external serial connection or the terminal program. This section describes the main methods used to interface to this object, along with the predefined constants that will make your program more readable.

Main Methods

start(bufaddr, bufsize)	**Start the overlay object.**

The **start** method requires the address and size of a **byte** array, which needs to be big enough to contain all of the data for all of the windows. The number of bytes required by each defined window is given by the following equation: **window requirement (bytes) = rows · columns + 10**

out(char)	Output a character to the overlay object.

The **out** method outputs one character to the overlay object. The character can be a command, a command parameter, or a character to display, depending upon the context of its use.

str(stringaddr)	Output a character string to the overlay object.

The **str** method, given the starting address of a zero-terminated character string, sends those characters to the overlay object. Because Spin strings cannot include zero-valued characters, an alternative "zero" is acceptable as an argument to the overlay commands: **$FF**. This value is predefined under several names in the overlay object: **_0**, **ZERO**, **CLRWIN**, **NO**, and **NONE**. Any of these names can be used to refer to this "virtual zero" in order to enhance a program's readability. Repeated below is one of the foregoing PBASIC code examples, followed by an equivalent program fragment in Spin:

```
i VAR Byte

SEROUT io,bd, [DEFWIN, 1, 8, 4]
SEROUT io,bd, [APNDSP, 5, SHO|1, 0, 0, SHODSP, 5]

FOR i = 0 TO 31
  SEROUT io,bd, [ESC, i]
NEXT
```

Assuming that **pr** is a reference to the overlay object, and that the overlay object's constants are locally defined, here is how the above code would be recast in Spin:

```
PUB start | i

  pr.str(string(DEFWIN, 1, 8, 4, APNDSP, 5, SHO|1, _0, _0, SHODSP, 5))
  repeat i from 0 to 31
    pr.out(ESC)
    pr.out(i)
```

Constants

The constants defined in the overlay object can always be referred to using the **object#constant** notation (e.g. **pr#ZERO**). But this can get awkward. It's more convenient simply to copy all the definitions into the object that uses them. Here's the complete list (which also serves to summarize all the commands):

```
CON

  'Commands recognized by "out" method.

  CLS     = $00 '( ) Clear the current window, and home cursor.
  HOME    = $01 '( ) Move cursor to home position.
  CRSRXY  = $02 '(col,row) Move cursor to col and row.
  CRSRLF  = $03 '( ) Move cursor to the left.
  CRSRRT  = $04 '( ) Move cursor to the right.
  CRSRUP  = $05 '( ) Move cursor up.
  CRSRDN  = $06 '( ) Move cursor down.
  SHODSP  = $07 '(dispno) Show display number dispno.
  BKSP    = $08 '( ) Erase prior character and move cursor left.
  TAB     = $09 '( ) Move cursor right to next column divisible by 8.
  LF      = $0A '( ) Linefeed. Scroll up if on bottom line.
  CLREOL  = $0B '( ) Clear to the end of the current line.
  CLRDN   = $0C 'Clear from cursor position to the end of the window.
  CR      = $0D '( )Carriage return. Scroll up if necessary.
  CRSRX   = $0E '(col) Move cursor to column col.
  CRSRY   = $0F '(row) Move cursor to row.
  DEFWIN  = $10 '(winno,cols,rows) Define a new window winno sized
                '  cols x rows. Make it the current window.
  USEWIN  = $11 '(winno) Change the current window to winno.
  CHGCLR  = $12 '(mask,transparent,fgnd,bkgd) Change current window color to
                '  mask, transparent, fgnd, and bkgd.
  SCROLL  = $13 '(offset) Set X (one-line) or Y (multi-line) scroll offset
                '  (0 - 15) for current window.
  SMSCRL  = $14 '(rate) Set smooth scrolling rate in current window to
                '  rate ms/scan line.
  WDWRAP  = $15 '(yn) Set word wrapping for current window: on (yn<>0)
                '  or off (yn==0).
  BLINK   = $16 '(yn) Set blinking for current window: on (yn<>0)
                '  or off (yn==0).
  CPYWIN  = $17 '(winno) Copy contents of winno to current window.
  APNDSP  = $18 '(disp,winno,x,y) Append window winno to display disp at
                '  location (x,y).
  MOVWIN  = $19 '(slot,x,y) Move window in slot to (x,y).
  SHOWIN  = $1A '(slot,yn) Show window in slot: yes (yn<>0) or no (yn==0).
  CHGWIN  = $1B '(slot,winno) Change window in slot to winno.
  PRESET  = $1C '(dispno,presetno) Create display dispno using preset
                '  presetno.
  SETTIM  = $1D '(yr,mo,day,hr,min,sec) Set the current time.
  MARK    = $1E '( )Return MARK to acknowledge reaching this point.
  ESC     = $1F '(char) Escape next character char (i.e. print as-is).

  CLRWIN  = $FF '( )Same as CLS where strings do not permit 0.
  NONE    = $FF 'Same as 0 when used as an argument.
  ZERO    = $FF 'Same as 0 when used as an argument.
  _0      = $FF 'Same as 0 when used as an argument.
  NO      = $FF 'Same as 0 when used as an argument.
  YES     = $01 'Canonical non-zero value used for binary choices.

  DBL     = $80 'OR with height and width arguments to get double-sized
                '  characters.
  SHO     = $40 'OR with window number to set visibility on.

  'Preset names.

  BIGWIN  = $01 '40 x 13 regular window.
  CREDITS = $02 'Vertically overscanned window with smooth scrolling.
  MARQUEE = $03 'Single row at bottom with smooth scrolling.
  HILO    = $04 'Single rows top and bottom.
  HILO2   = $05 'Dual rows top and bottom.
  CROSS   = $06 'Single cross-shaped cursor in middle of 40 x 13 screen.
  BOX     = $07 'Single box-shaped cursor in middle of 40 x 13 screen.
  DOT     = $08 'Single dot cursor in middle of 40 x 13 screen.

  ' Window type names.

  REGWIN   = 0  'Regular window.
  HMS24    = 1  'Time window: 23:59:59
  HMS12    = 2  'Time window: 12:59:59 pm
  YMD      = 3  'Date window: 2099-12-31
  MDY      = 4  'Date window: 12/31/2099
  DMY      = 5  'Date window: 31-12-2099
  YMDHMS24 = 6  'Date/time window: 2099-12-31 23:59:59
  MDYHMS12 = 7  'Date/time window: 12/31/2099 12:59:59 pm
  DMYHMS12 = 8  'Date/time window: 31-12-2099 12:59:59 pm
```