roControlPort

ON THIS PAGE

Firmware Version 7.1

- Version 7.1
- Version 7.0
- Version 6.2
- Version 6.1
- Previous Versions

This object provides support for the BP200/BP900 USB button boards, GPIO ports, and side buttons on the BrightSign player. Button presses are returned as *roControlUp* and *roControlDown* events. The object is used to configure output levels on the I/O connector and monitor inputs. Typically, LEDs and buttons are attached to the GPIO connector on the BrightSign player or the BrightSign Expansion Module.

Object Creation: The *roControlPort* object is created with a single parameter that specifies the port being used.

```
CreateObject("roControlPort", port As String)
```

The port parameter can be one of the following:

- `BrightSign`: Specifies the onboard GPIO connector (including the SVC (GPIO12) button).
- `Expander-GPIO`: Specifies the DB-25 connector on the BrightSign Expansion Module. If no BrightSign Expansion module is attached, then object creation will fail and Invalid will be returned.
- `Expander-<n>-GPIO`: Specifies a USB-to-GPIO device connected to the player. Multiple USB-to-GPIO devices can be controlled using separate *roControlPort* instances: The first device corresponds to `"Expander-0-GPIO"`, the second to `"Expander-1-GPIO"`, etc.
- `Expander-DIP`: Specifies the eight DIP switches on the BrightSign Expansion Module. If no BrightSign Expansion module is attached, then object creation will fail and Invalid will be returned.

Note

Hot-plugging the BrightSign Expansion Module is not supported.

- `Touchboard-<n>-GPIO`: Retrieves events from the specified BP200/BP900 button board. Events are handled in the same manner as events from the BrightSign port.
- `Touchboard-<n>-LED-SETUP`: Sets various LED output options for the specified BP200/BP900 button board.
- `Touchboard-<n>-LED`: Sets the bits for each button on the specified BP200/BP900 button board. The bits indicate whether the associated LED should be on or off.

> **Note**
>
> Since multiple BP200/BP900 button boards can be connected to a player simultaneously, the <n> value specifies the port enumeration of each board. This value corresponds to the `<raw>` or `<fid>` value returned by the *roDeviceInfo.GetUSBTopology()* method. An unspecified enumeration value is synonymous with a button board with an enumeration value of 0 (e.g. Touchboard-GPIO and Touchboard-0-GPIO are identical).

ifControlPort

> **Note**
>
> The "button" numbers described below are not the same as GPIO "pin" numbers: Some pins act as power supply or ground, so they are not included in the button numbering scheme. See the hardware manual associated with your player model to view a mapping of buttons to pins.

GetVersion() As String

Returns the version number of the firmware (either the main BrightSign firmware or the BrightSign Expansion Module firmware) responsible for the control port.

EnableOutput(button As Integer) As Boolean

Marks the specified button as an output. If an invalid button number is passed, `false` will be returned. If successful, the function returns `true`. The output will be driven high or low depending on the current output state of the pin.

> **Tip**
>
> See here for a table of pins and corresponding buttons for the onboard GPIO connector.

EnableInput(button As Integer) As Boolean

Marks the specified button as an input. If an invalid button number is passed, `false` will be returned. If successful, the function returns `true`. The button will be tri-stated and can be driven high or low externally.

EnableAlternateFunction(button As Integer, pin_function As String) As Boolean

Enables an alternate function on a GPIO button. This method applies to the onboard GPIO connector and is currently supported on the XTx44, XTx43, XDx34, XDx33, HDx23, and HO523 models.

The first argument specifies the GPIO button number (between 0 and 7). The second argument specifies the alternate function setting; the following table outlines the possible alternate setting for each pin:

GPIO Pin	Button Number	Alternate Function
3	0	`"serial1"` (Rx)
4	1	`"irin1"`
5	2	`"irout"` (HDx23, HO523 only)
6	3	N/A
9	4	`"serial0"` (Rx - console port)*
10	5	`"serial0"` (Tx)*
11	6	`"serial1"` (Tx)

12	7	N/A

*Models that do not have a 3.5mm serial port (e.g. HD223, XD233) do not support serial port 0.

> **Note**
>
> To revert a GPIO button to its primary function, specify the `pin_function` as "gpio".

GetWholeState() As Integer

Returns the state of all the inputs attached to the control port as bits in an integer. Individual buttons can be checked using binary operations, although it is normally easier to call `IsInputActive()` instead.

IsInputActive(button As Integer) As Boolean

Returns the state of the specified input. If the button is not configured as an input, then the result is undefined.

SetWholeState(state As Integer) As Boolean

Specifies the desired state of all outputs attached to the control port as bits in an integer. The individual buttons can be set using binary operations, although it is normally easier to call `SetOutputState()` instead.

Example

```
port = CreateObject("roControlPort", "BrightSign")
gpio1 = 2 '2^1
gpio3 = 8 '2^3
gpio5 = 32 '2^5
gpio7 = 128 '2^7
port.SetWholeState(gpio1 + gpio2 + gpio5 + gpio7) 'turns on ports 1, 3, 5,
and 7
```

SetOutputState(button As Integer, level As Boolean) As Boolean

Configures the output of the specified pin, which can be either "off" (0) or "on" (1). If the button is not configured as an output, the resulting level is undefined. This method can also be used to configure LED output behavior on BP200/B900 button boards; see the **BP200/BP900 Setup** section below for more details.

SetOutputValue(offset As Integer, bit-mask As Integer) As Boolean

Configures a button on a BP200/BP900 button board. This method can only be used when the *roControlPort* object is instantiated with the `Touchboard-<n>-LED-SETUP` or `Touchboard-<n>-LED` parameter. See the **BP200/BP900 Setup** section below for more details.

SetOutputValues(values As roAssociativeArray) As Boolean

Configures buttons on a BP200/BP900 button board. This method can only be used when the *roControlPort* object is instantiated with the `Touchboard-<n>-LED-SETUP` or `Touchboard-<n>-LED` parameter. See the **BP200/BP900 Setup** section below for more details.

GetProperties() As roAssociativeArray

Returns an associative array of values related to the attached BP200/BP900 button board, including hardware, header, and revision. This method can only be used with an *roControlPort* instantiated with the `Touchboard-<n>-GPIO` parameter.

SetPulseParams(parameters As roAssociativeArray) As Boolean

Specifies a period of time, as well as the time slices within that period, for pulsing GPIO LEDs. These properties are applied to all GPIO outputs. This method is passed an associative array with the following parameters:

- `milliseconds`: An integer specifying the time period (in ms) for pulsing
- `slices`: An integer specifying the number of divisions within the milliseconds time period: For example, a 500ms time period with slices: 2 is divided into two 250ms slices.

SetPulse(button As Integer, bit-field As Integer) As Boolean

Sets the off/on bit field for a particular GPIO. Use the `slices` parameter of the `SetPulseParams()` method to determine the number of bits in the bit field. For example, specifying `milliseconds:500, slices:2`, and a bit field of `10` will cause the button to turn on every other 250 millisecond period.

RemovePulse(button As Integer) As Boolean

Removes the specified GPIO from the set affected by the pulse.

ifMessagePort

SetPort(port As Object)

Posts messages of type *roControlUp* and *roControlDown* to the attached message port.

ifUserData

SetUserData(user_data As Object)

Sets the user data that will be returned when events are raised.

GetUserData() As Object

Returns the user data that has previously been set via `SetUserData()`. It will return Invalid if no data has been set.

ifIdentity

GetIdentity() As Integer

Returns an identity value that can be used to associate *roControlUp* and *roControlDown* events with this control port.

> **Note**
>
> The *ifIdentity* interface has been deprecated. We recommend using the *ifUserData* interface instead.

This example script applies timed pulses to a set of GPIOs:

```
' set up button 2 and 3 to flash at 2Hz (i.e. on & off twice in a second)
in an alternating ' fashion.

gpioPort = CreateObject("roControlPort", "BrightSign")

gpioPort.EnableOutput(2)
gpioPort.SetOutputState(2, true)

gpioPort.EnableOutput(3)
gpioPort.SetOutputState(3, true)

' set up pulse to have two time slices of 250ms each.
gpioPort.SetPulseParams({ milliseconds: 500, slices: 2 })

' button 2 will have slice 1 on and slice 2 off.
gpioPort.SetPulse(2, &h01)

' button 3 will have the reverse of button 2.
gpioPort.SetPulse(3, &h02)

' wait for a bit.
sleep(10000)

' stop pulsing on button 2.
gpioPort.RemovePulse(2)
```

This example script enables various alternate functions on the GPIO:

```
c = CreateObject("roControlPort", "BrightSign")

'Enable serial port 1 on the GPIO.

c.EnableAlternateFunction(0, "serial1")
c.EnableAlternateFunction(6, "serial1")

s1 = CreateObject("roSerialPort", 1, 115200)
s1.SendLine("This is serial port 1")

mp = CreateObject("roMessagePort")
s1.SetLineEventPort(mp)
? wait(10000, mp)

'Switch serial port 0 from the 3.5mm serial port to the GPIO.
'[Note: it is advised use telnet/ssh or a script when testing this]

c.EnableAlternateFunction(4, "serial0")
c.EnableAlternateFunction(5, "serial0")

s = CreateObject("roSerialPort", 0, 115200)
```

```
s.SendLine("Hello on the console?")

mp = CreateObject("roMessagePort")
s.SetLineEventPort(mp)
? wait(10000, mp)

'Restore normal operation on serial port 0.

c.EnableAlternateFunction(4, "gpio")
c.EnableAlternateFunction(5, "gpio")

'Enable IR input on the GPIO.

c = CreateObject("roControlPort", "brightsign")
? c.EnableAlternateFunction(1, "irin1")

nexus_encodings = [ "NEC", "NEC32" ]
ir_gpio = CreateObject("roIRReceiver", { source: "GPIO", encodings:
nexus_encodings })

mp = CreateObject("roMessagePort")
ir_gpio.SetPort(mp)

m = wait(10000, mp)

'Enable IR output on the GPIO (HDx23, HO523 only--the XT/XD models have a
dedicated 3.5mm IR socket)

c.EnableAlternateFunction(2, "irout")

ir = CreateObject("roIRTransmitter", { destination: "GPIO"} )
ir.Send("NEC32", &H12345)
```

BP200/BP900 Setup

To send a configuration to the BP200/BP900 button board, instantiate *roControlPort* with the `Touchboard-<n>-LED-SETUP` parameter and call the `SetOuputValue()` method. This method accepts two integers: the first integer specifies one of three command types (offsets); the second integer is a bit field consisting of 32 bits.

- **Offset 0**: Configures the button board using a bit field that is split into four bytes of eight bits each. Each byte is a separate part of the configuration. In the script, these bytes need to be listed from right to left in hex value (i.e. Byte 1 + Byte 2 + Byte 3 + Byte 4).
    - Byte 1: Specifies the configuration type for the button board. Currently, the only configuration type is for LED output, which is specified with the value &hA0.
    - Byte 2: The button number(s) that will be configured. Buttons are numbered beginning from 1. The value is set to 0 (&h00) if this command is not required.
    - Byte 3: The LED bit-field configuration. This value specifies how many on/off bits should be used (up to 32 bits) when `SetOutputValue()` is called on a `Touchboard-<n>-LED` instance (see the **BP200/BP900 LED Output** section below for details). Set the value to 0 (&h00) if this command is not required (the bit field will be set to eight bits by default).
    - Byte 4: This value is currently always set to 0 (&h00).
- **Offset 1**: Disables buttons on the button board according to values in the bit field. Each button is disabled individually by setting bits 0-10: For example, passing the hex value &h00000008 will disable button 4 only.
- **Offset 2**: Disables LEDs on the button board according to values in the bit field. Each LED is disabled individually by setting bits 0-10: For example, passing the hex value &h00000080 will disable the LED on button 8 only.

Note

Disabling a button LED will not automatically disable the button itself (and vice-versa). To disable both the button and the LED, make separate SetOutputValue() calls for Offset 1 and Offset 2.

BP200/BP900 LED Output

To control the behavior of individual button LEDs, instantiate *roControlPort* with the `Touchboard-<n>-LED` parameter, then pass per-LED bit fields to the `SetOutputValue()` method. This method accepts two integers: the first integer specifies the button number (0-11), while the second integer uses a bit field to specify the on/off behavior of the button LED. The size of the bit field (up to 32 bits) is determined with the Offset 0 – Byte 3 value described in the section above.

Each bit specifies the on/off behavior of a single cycle, and the BP200/BP900 button boards run at approximately 11Hz. For example, if you want an LED to cycle on every other second, you would set the Offset 0 – Byte 3 value to &h16 (22 bits) and the bit field itself to &h3FF800 (00000000000111111111111).

This example script sets a BP900 to “twinkle” by turning off each button LED at a different point in the cycle:

```
led=CreateObject("roControlPort", "TouchBoard-0-LED")
led_setup=CreateObject("roControlPort", "TouchBoard-0-LED-SETUP")
led_setup.SetOutputValue(0, &h000B00A0)
led.SetOutputValue(0, &h07fe)
led.SetOutputValue(1, &h07fd)
led.SetOutputValue(2, &h07fb)
led.SetOutputValue(3, &h07f7)
led.SetOutputValue(4, &h07ef)
led.SetOutputValue(5, &h07df)
led.SetOutputValue(6, &h07bf)
led.SetOutputValue(7, &h077f)
led.SetOutputValue(8, &h06ff)
led.SetOutputValue(9, &h05ff)
led.SetOutputValue(10, &h03ff)
```