BBXML User's Guide

Version: 1.3
Author: Darin Franklin

BBXML is an XML interface for controlling the scrolling LED signs made by Adaptive Micro Systems, including the BetaBrite and Alpha models.

Adaptive provides complete documentation for their Alpha Sign Communication Protocol, which allows the sign to be controlled from a computer's serial port. The Alpha Sign Protocol offers complete access to all of the sign's features, but it is a rather arcane message format. The BBXML interface makes this easier, by providing access to all of the commands in the protocol through simple XML messages. BBXML works with all BetaBrite and Alpha signs that support the Alpha Sign Protocol.

This document contains setup instructions, a usage summary, and a complete command reference.

Download

The package is available at https://github.com/darinfranklin/bbxml.

Required Components

The core component of BBXML is a single XSLT file, `alphasign.xsl`. That is really all you need from this package. If you have an old sign, you might need to use `bb3to1` or `bb3to1.c`, as explained later, to postprocess the XSLT output.

To run the examples in this document, you need an XSLT processor that executes from the command line and writes to standard output. For this purpose, I recommend xsltproc, which is part of the libxslt package.

Some of the example and utility scripts require Perl, but Perl is not required for general usage.

The examples were written for Linux, but Linux is not required. To make this work on your system, you need to know how to configure the serial port, how to write to the serial port, and how to run an XSLT processor.

Installation

Unpack the `bbxml_1.3.tar.gz` file into `/opt`, or any convenient location.

Initial Setup

Wiring

You need to connect the sign to your computer's RS-232 serial port with a properly-wired cable. If your sign didn't come with a cable, you can purchase one from Adaptive (it's included with their Windows software), or you can make one yourself. See TechMemo 02-0010 on Adaptive's site for pinouts and wiring details.

If you want to make a cable, I recommend using a modular adapter and silver satin cable. (Those links are just examples; I don't know anything about that site.) You need to connect the TxD pin (3) on the computer to the RxD pin (3) on the sign, and the RxD pin (2) on the computer to the TxD pin (4) on the sign. You also need to connect the signal ground pins to each other (pin 5 on the computer to pin 6 on sign). Make sure that no stray wire ends are touching. The Alpha signs use those "extra" wires for RS-485. If they get shorted, all kinds of bad things will happen.

Environment Variables

The included scripts assume that the BBXML package is installed at `/opt/bbxml`, and that the sign is connected to the port at `/dev/betabrite`. You can override these values by setting the the `BBXML_HOME` and `BBXML_PORT` environment variables. For example:

```
export BBXML_HOME=/home/darin/opt/bbxml
export BBXML_PORT=/dev/ttyS2
```

You will probably also want to add the bin directory to your PATH

```
export PATH=$PATH:/opt/bbxml/bin
```

Serial Port Setup

First, create a symbolic link, named `/dev/betabrite`, which points to the serial port that your sign is connected to. For example, if your sign is on `/dev/ttyS0`, create the link like this.

```
ln -s /dev/ttyS0 /dev/betabrite
```

If you do not want to use "`/dev/betabrite`" as the port name, then you must set the BBXML_PORT environment variable, as described previously.

Before you can send anything to the sign, you need to configure the serial port to use 9600 baud, 7 bits, even parity, and 1 stop bit. (See Adaptive's docs for other valid configurations.) Also, disable postprocessing and line ending translation, if necessary. On Linux, use `stty` to this:

```
stty -F /dev/betabrite 9600 -opost -ocrnl -onlcr cs7 parenb -parodd clocal
```

As a shortcut, you can execute the stty command above by running this command instead:

```
bbxml --initport
```

If you are running Windows, then you should be able to find a dialog in Control Panel to change these settings.

Usage Summary

Command Line

Write some commands in a file, `commands.xml`, and then send it to the sign like this:

```
bbxml commands.xml
```

The rest of this section explains the gory details.

How It Works

The `alphasign.xsl` file contains the all of the rules that the XSLT processor uses to transform your XML document into an Alpha Sign Protocol message.

This is the basic procedure for sending commands to the sign.

1. Write an XML file with the commands that you want to send.
2. Transform it into an Alpha Sign Protocol message using XSLT.
3. Write the Alpha Sign Protocol message to the serial port.

The `bbxml` script does steps 2 and 3 for you.

Creating the XML File

Write the commands in a text file. Here is an example, `commands.xml`:

```
<alphasign>
  <text label="0">BetaBrite</text>
</alphasign>
```

Transforming into Alpha Sign Protocol

Use the XSLT processor to transform your XML document with the `alphasign.xsl` stylesheet.

```
xsltproc alphasign.xsl commands.xml > commands.txt
```

The output is the 3-byte flavor of the Alpha Sign Protocol.

```
cat commands.txt
```

```
_01Z00_02A0BetaBrite_04
```

Writing to the Sign

To send the commands to the sign, simply `cat` it to the device file:

```
cat commands.txt > /dev/betabrite
```

If all is well, you should see "BetaBrite" scrolling across the sign. If that didn't work, don't worry. Read the next section about the 3-byte protocol.

3-Byte vs. 1-Byte Protocol

The original version of the Alpha Sign Protocol used ASCII control characters, such as SOH (0x01, '^A'), for commands. This is known as the 1-byte protocol. Later versions allowed for the commands to be written in printable ASCII characters instead of control codes. The 2-byte ASCII printable format uses "]" as an escape character, followed by a single character to indicate the command. The 3-byte format uses "_" as the escape character, followed by a two digit hex value.

The `alphasign.xsl` transform outputs messages in the 3-byte format. If your sign has an old firmware, such as my BetaBrite 1040 ("1040-4402a EZII", 1995), it might only be able to read the 1-byte protocol. Here is a simple test to see if your sign can read the 3-byte protocol.

```
echo "_01Z00_02A0hello_04" > /dev/betabrite
```

If you see "hello" on your sign, then the 3-byte protocol works for you. If it didn't work, try using the included `bb3to1` script to convert the message to 1-byte format.

```
echo "_01Z00_02A0hello_04" | bb3to1 > /dev/betabrite
```

That should work for every sign. If it didn't, check your wiring and port configuration and try again.

Do this to clear the message:

```
echo "_01Z00_02A0_04" > /dev/betabrite
```
or
```
echo "_01Z00_02A0_04" | bb3to1 > /dev/betabrite
```

More about bb3to1

If you convert the commands.txt file from above with bb3to1, you can see the difference in message format.

```
bb3to1 commands.txt > commands.1.txt
cat -v commands.1.txt
```

```
\0\0\0\0\0^AZ00^BAHBetaBrite^D
```

You may be wondering why we don't just output the 1-byte format from XSLT, since all signs understand that format. We can't do that because the XML 1.0 spec does not allow ASCII control characters (`#x00-#x1F`), except for TAB, CR, and LF. Therefore, we write the 3-byte ASCII printable version from XSLT and then convert it with `bb3to1`, if necessary.

If, for some reason, you can't use Perl, I have included a version of `bb3to1` written in C: `bb3to1.c`. Compile it and then use it in place of the Perl script.
```
gcc -o bb3to1 bb3to1.c
```

XML Command Reference

Alpha Sign Protocol Basics

There are just a few things that you need to know about how the sign works before you can start writing XML commands.

Files

The sign holds data in memory locations called "files". There are three file types: TEXT, STRING, and DOTS. TEXT files hold the messages that the sign displays. STRING files are like variables that you can include in TEXT files. DOTS files are also like variables, but contain

graphics instead of characters.

Labels

Each file has a 1-character label. The three file types are kept in separate name spaces, so you can use the same label for a TEXT, a STRING, and a DOTS file without conflict. Valid file labels are listed in Appendix A of the protocol documentation.

There is one special TEXT file, known as the Priority TEXT file, which has the label, "`0`". Any message sent with the "`0`" label will be displayed immediately and will continue to be displayed until you send a blank message with the "`0`" label. The "`0`" file is always allocated 125 bytes, and the size cannot be changed.

Protocol Formats

The Alpha Sign Protocol has three flavors: 3-byte, 2-byte, and 1-byte. The XML interface outputs the 3-byte format. You can convert 3-byte format into 2-byte or 1-byte by using the scripts, `bb3to2` and `bb3to1`.

XML Message Structure

The complete BBXML message format is defined in `alphasign.dtd`.

The root element of every message is `alphasign`.

```
<alphasign>
  <!-- commands go here -->
</alphasign>
```

The `alphasign` element has two optional attributes: `typeCode` and `signAddress`. The `typeCode` attribute is a 1-character code that specifies type of sign that you want to communicate with. Refer to the Alpha protocol documentation for the list of Type Code values. If you omit this attribute, it will default to "`Z`", the code for "all signs".

The `signAddress` attribute is the 2-character uppercase hexadecimal address of the sign (00 - FF). If you omit this attribute, it defaults to "`00`", which is the only address that BetaBrite signs respond to.

```
<alphasign typeCode="^" signAddress="00">
  <!-- commands go here -->
</alphasign>
```

For BetaBrite, just omit both attributes.

Since this is XML, you have to escape the characters, "&", "<", and ">".

- & => &
- < => <
- > => >

For example, if you want to use typeCode of "`&`":

```
<alphasign typeCode="&amp;">
  <!-- commands go here -->
</alphasign>
```

Command Index

Inside the `alphasign` element, include one or more of these commands:

- memoryConfig
- text
- string
- dots
- sequence
- daySchedule Table
- timeScheduleTable
- timeOfDay
- dayOfWeek
- beep
- softReset
- dimmingMode
- clearSerialErrorStatusRegister
- calendarDate
- counterConfig
- counterValue
- temperatureDisplay
- temperatureOffset
- signAddress
- xyTextMode

- speakerMode
- timeFormat
- "read" commands

```
<alphasign>
  <text label="0">BetaBrite</text>
</alphasign>
```

Configuring the Memory

To clear the memory and reset it to the factory default configuration, call `memoryConfig` without any child elements.

```
<alphasign>
  <memoryConfig/>
</alphasign>
```

After the memory has been reset, only the "`A`" and "`0`" TEXT files are usable. If you want to use more than that, you have to reserve memory for the files first. Do that by adding child elements to `memoryConfig`, either `textConfig`, `stringConfig`, or `dotsConfig`.

```
<alphasign>
  <memoryConfig>
    <textConfig label="A" size="768"/>
    <textConfig label="D" size="768" start="always" stop="never"/>
    <textConfig label="m" size="512" start="always" stop="never" locked="true"/>
    <stringConfig label="D" size="11"/>
    <stringConfig label="t" size="3"/>
    <dotsConfig label="A" height="7" width="9" colors="8" locked="false"/>
  </memoryConfig>
</alphasign>
```

This also clears the sign's memory, so you have to configure all files at once. You cannot add new files without wiping out the existing ones.

The `label` attribute is required for all three file types. It is the 1-character name of the file that you are configuring. You can name a STRING file, a TEXT file, and a DOTS file all with the same label without conflict.

Text Memory Config

The `size` attribute specifies how many bytes to reserve for a TEXT file. It is required and must contain an integer value.

The `start` and `stop` attributes specify the initial time schedule for the TEXT file. See timeScheduleTable for the list of possible values. These are optional and default to "`always`" and "`never`", respectively.

The `locked` attribute is optional and defaults to "`false`". If `locked` is set to "`true`", then you will not be able to change the file using the remote control.

```
<alphasign>
  <memoryConfig>
    <textConfig label="D" size="768" start="05:00" stop="15:30" locked="true"/>
  </memoryConfig>
</alphasign>
```

String Memory Config

The `size` attribute specifies how many bytes to reserve for the STRING file. It is required and must contain an integer value.

```
<alphasign>
  <memoryConfig>
    <stringConfig label="D" size="11"/>
  </memoryConfig>
</alphasign>
```

Dots Memory Config

The `height` and `width` attributes specify the number of pixels to allocate for the DOTS file. Both attributes are required and must contain integer values.

The `colors` attribute sets the color status for the DOTS file. It is optional and defaults to "`8`". The only valid values are `1`, `3`, and `8`, for monochrome, 3-color, and 8-color modes, respectively.

The `locked` attribute is optional and defaults to "`false`". If `locked` is set to "`true`", then you will not be able to change the file using the

remote control.

```
<alphasign>
  <memoryConfig>
    <dotsConfig label="A" height="7" width="9" colors="8" locked="false"/>
  </memoryConfig>
</alphasign>
```

TEXT files

A TEXT file is simply a message that you want to display on the sign. To send a TEXT file, use the `text` element.

```
<alphasign>
  <text label="0">The quick brown fox jumps over the lazy dog.</text>
</alphasign>
```

The required `label` attribute specifies the 1-character file name.

To clear the contents of a TEXT file, send an empty message.

```
<alphasign>
  <text label="0"/>
</alphasign>
```

Breaking up Long Lines

Sometimes it's necessary to break up long lines of text in order to make the XML file more readable. If you do this, you need to ensure that the extra line breaks are not included in the text that goes to the sign. As a first attempt, you might try something like this:

```
<alphasign>
  <text label="0">
    The quick brown fox
    jumps over the lazy dog.
  </text>
</alphasign>
```

Run that through the processor and see what happens.

```
xsltproc alphasign.xsl dog.xml
```

```
_01Z00_02A0
    The quick brown fox
    jumps over the lazy dog.
  _04
```

That's not quite what you wanted. If you sent that to the sign, you would see a lot of extra spaces in front of "The" and after "fox". To avoid this, use the `msg` element to keep the extra spaces from showing up on the sign. The `msg` element encloses significant text within XML tags. The XML parser then treats the extra spaces as insignificant whitespace.

```
<alphasign>
  <text label="0">
    <msg>The quick brown fox </msg>
    <msg>jumps over the lazy dog.</msg>
  </text>
</alphasign>
```

That's better:

```
xsltproc alphasign.xsl dog.xml
```

```
_01Z00_02A0The quick brown fox jumps over the lazy dog._04
```

Modes

The `text` element has an optional `mode` child element. The `display` attribute is required, and specifies the display effect. The optional `position` attribute indicates the vertical position for multiline signs.

```
<alphasign>
  <text label="0"><mode display="rollUp"/>BetaBrite<mode display="rollDown"/></text>
</alphasign>
```

If no `mode` element is included, the sign defaults to "automode." As shown in the example above, you can include a mode after the text as well.

The sign uses that effect to clear the display before showing the next message.

Display Effect

Here are the valid values for `mode`'s `display` attribute. Items in the third column produce animated graphics. Not all signs support all effects.

- `hold`
- `rotate`
- `flash`
- `scroll`
- `compressedRotate`
- `explode`
- `clock`
- `twinkle`
- `sparkle`
- `snow`
- `interlock`
- `switch`
- `slide`
- `cycleColors`
- `spray`
- `starburst`
- `automode`
- `wipeUp`
- `wipeDown`
- `wipeLeft`
- `wipeRight`
- `wipeIn`
- `wipeOut`
- `rollUp`
- `rollDown`
- `rollLeft`
- `rollRight`
- `rollIn`
- `rollOut`
- `welcome`
- `slotMachine`
- `newsFlash`
- `trumpet`
- `thankYou`
- `noSmoking`
- `drinkAndDrive`
- `runningAnimal`
- `fish`
- `fireworks`
- `turboCar`
- `balloon`
- `cherryBomb`

Multiline Position

For multiline signs, you may also specify the vertical position of the text with the optional `position` attribute. The values for `position` are:

- `middle` - text is centered vertically
- `top` - text starts on first line of sign
- `bottom` - text starts on the line immediately following the line most recently marked as "top"
- `fill` - text is centered vertically, filling all available lines
- `left` - text begins on the left side of the sign, filling all but 1 line. (Alpha 3.0 protocol only)
- `right` - text begins on the left side of the sign, filling all but 1 line. (Alpha 3.0 protocol only) (Yes, the doc says "text begins on the *left*". I suspect they meant "right", but I can't be sure.)

The default value is `middle`.

```
<alphasign>
  <text label="0"><mode display="hold" position="top">BetaBrite</text>
</alphasign>
```

Multiple Modes

You can include multiple modes in a single TEXT file. Simply insert `mode` elements within the message text, or intersperse `mode` and `msg` elements.

```
<alphasign>
  <text label="A"><mode display="rollDown"/>One<mode display="rollUp"/>Two<mode display="starburst"/></text>
</alphasign>
```

```
<alphasign>
  <text label="A">
    <mode display="rollDown" position="top"/>
    <msg>Top</msg>
    <mode display="starburst" position="top"/>
    <mode display="rollUp" position="bottom"/>
    <msg>Bottom</msg>
    <mode display="rotate" position="bottom"/>
  </text>
</alphasign>
```

Formatting Codes

TEXT files can contain formatting codes to change the color and style of the text or to insert special characters. Not all codes are supported by the BetaBrite; some of them work on Alpha signs only. See the Alpha Sign Protocol documentation for details.

Colors

- red
- dimred
- orange
- brown
- yellow
- amber
- green
- dimgreen
- rainbow1
- rainbow2
- colormix
- autocolor

When you insert a color code into the message, all of the characters that follow will be in that color. If you don't specify a color, the sign defaults to autocolor.

```
<alphasign>
  <text label="0"><green/>The quick <brown/>brown <green/>fox...</text>
</alphasign>
```

Special Characters and Data

- `CR` - insert line break
- `FF` - insert page break
- `extendedChar`, attribute: `offset` - insert special character
- `time` - insert time of day
- `date`, attribute: `format` - insert date
- `callString`, attribute: `label` - insert STRING file
- `callDots`, attribute: `label` - insert DOTS file
- `callLargeDots`, attribute: `label` - insert LARGE DOTS PICTURE file

```
<alphasign>
  <text label="0">
      <mode display="rotate"/>
      <msg>Date: <date format="MM/DD/YY"/><CR/></msg>
      <msg>Time: <time/><CR/></msg>
      <msg>Temp: <callString label="t"/><extendedChar offset="49"/>F<CR/></msg>
      <msg>Picture: <callDots label="i"/></msg>
  </text>
</alphasign>
```

Use `CR` to insert a line break. Use `FF` to insert a page break (only meaningful for multi-line signs). Use `time` to insert the current time.

Use `extendedChar` and its required `offset` attribute to insert a special character. Refer to Appendix G in the Alpha Sign Protocol documentation for a list of extended characters and their offset codes. For example, Appendix G says that the control code for the upside-down question mark is "08H + 48H". Therefore, specify "48" as the offset value: `<extendedChar offset="48"/>`. The example above inserts the degree symbol, code "49".

Use `date` and its required `format` attribute to insert the current date. (BetaBrite signs don't support `date` because they don't have an internal calendar.) The date format attribute must be one of these 10 values.

- `MM/DD/YY`
- `DD/MM/YY`
- `MM-DD-YY`
- `DD-MM-YY`
- `MM.DD.YY`
- `DD.MM.YY`
- `MM DD YY`
- `DD MM YY`
- `MMM.DD, YYYY`
- `day`

Use `callString` and its required `label` attribute to insert the contents of a STRING file.

Use `callDots` and its required `label` attribute to insert the contents of a DOTS file.

The `callLargeDots` command is not supported in this version.

Speeds

- `noHold`
- `speed1` *(slowest)*
- `speed2` *(slower)*
- `speed3` *(slow)*
- `speed4` *(default)*
- `speed5` *(fast)*
- `speedControl`, attribute: `minutes`, `seconds` or `deciseconds`

```
<alphasign>
  <text label="0"><mode display="hold"/><speed5/>The quick brown fox...<noHold/></text>
</alphasign>
```

The `noHold` speed causes the individual "pages" of the message to display without pausing. Insert it at the beginning of the message to eliminate pause that normally occurs.

```
<alphasign>
  <text label="0"><mode display="hold"/><noHold/><speed5/>The<CR/>quick<CR/>brown<CR/>fox</text>
</alphasign>
```

The `speedControl` code only works on certain Alpha signs. Use either the `seconds` attribute or the `deciseconds` attribute, depending on which one your sign supports. See the protocol doc for more details.

```
<alphasign>
  <text label="0"><mode display="rotate"/><speedControl deciseconds="5"/>zooooooom</text>
</alphasign>
```

Styles

Like colors, style control codes turn on a particular format. Text that follows a style control code will display in that style, until you change it with another control code. Not all signs support all styles.

Styles

- `wideModeOn`
- `wideModeOff`
- `doubleHighModeOn`
- `doubleHighModeOff`
- `trueDescendersModeOn`
- `trueDescendersModeOff`
- `fixedWidthModeOn`
- `fixedWidthModeOff`
- `flashModeOn`
- `flashModeOff`
- `wideOn`
- `wideOff`
- `doubleWideOn`
- `doubleWideOff`
- `trueDescendersOn`
- `trueDescendersOff`
- `fixedWidthOn`
- `fixedWidthOff`
- `doubleHighOn`
- `doubleHighOff`
- `fancyOn`
- `fancyOff`
- `shadowOn`
- `shadowOff`

Character Sets

- `standard5`
- `slim5`
- `stroke5`
- `wide5`
- `wideStroke5`
- `custom5`
- `standard7`
- `slim7`
- `stroke7`
- `wide7`
- `custom7`
- `fancy7`
- `slimFancy7`
- `wideFancy7`
- `shadow7`
- `shadowFancy7`
- `strokeFancy7`
- `wideStroke7`
- `wideStrokeFancy7`
- `standard10`
- `custom10`
- `custom15`
- `fullHeightFancy`
- `fullHeightStandard`
- `auxPortOn`
- `auxPortOff`

On the BetaBrite, you can combine `wideModeOn` with `fancy7` to achieve a double-wide effect. You can also combine `fixedWidthModeOn`

with `fancy7` to get an unreadable overlapping effect.

```
<alphasign>
  <text label="0"><mode display="rotate"/>
    <msg>The </msg>
    <msg><orange/>quick </msg>
    <msg><amber/>brown </msg>
    <msg><dimred/>fox </msg>
    <msg><green/><wideFancy7/>jumps</msg>
    <standard7/><autocolor/>
    <msg> over the </msg>
    <msg><wideModeOn/>L<wideModeOff/><standard5/>azy </msg>
    <standard7/><rainbow3/>
    <msg><wideModeOn/>D<wideModeOff/><standard5/>og</msg>
  </text>
</alphasign>
```

STRING files

To send a STRING file, use the `string` element.

```
<alphasign>
    <string label="D">weasel</string>
</alphasign>
```

To include the value of a STRING file in a TEXT file, use `callString` inside the text.

```
<alphasign>
  <text label="0"><mode display="hold">
    <msg>The quick brown fox </msg>
    <msg>jumps over the lazy <callString label="D"/>.</msg>
  </text>
</alphasign>
```

The advantage of STRING files is that the sign does not flicker or pause when you send them. STRING files are good for storing small pieces of information that fit into the bigger template of a TEXT file.

On some signs, you can also include the commands for speeds, character set selection, color selection (excluding `rainbow1` and `rainbow2`), or any of the following:

- CR
- time
- wideModeOn
- wideModeOff
- fixedWidthModeOn
- fixedWidthModeOff

```
<alphasign>
    <string label="D"><wideModeOn/>T<wideModeOff/>ime<CR/><time/></string>
</alphasign>
```

DOTS files

Use the `dots` element to define arbitrary graphics. Each `row` element defines one row of pixels (LEDs). Each digit in a row defines the color of a pixel.

```
<alphasign>
  <!-- a musical note -->
  <dots label="A">
    <row>000011100</row>
    <row>000010110</row>
    <row>000010011</row>
    <row>000010110</row>
    <row>011010000</row>
    <row>111100000</row>
    <row>011000000</row>
  </dots>
</alphasign>
```

The colors are defined as follows.

- `0` - pixel off

- `1` - red
- `2` - green
- `3` - amber
- `4` - dim red
- `5` - dim green
- `6` - brown
- `7` - orange
- `8` - yellow

Like STRING files, DOTS files do not display directly, but must be included inside TEXT files. Insert `callDots` inside the text message.

```
<alphasign>
  <text label="0"><mode display="rotate"/>
    <callDots label="A"/>
    <callDots label="A"/>
    <callDots label="A"/>
    <msg>Now Playing: Jumps Over the Lazy Dog (The Quick Brown Fox)</msg>
  </text>
</alphasign>
```

Remember that you must use `memoryConfig` to configure the DOTS file size before sending the DOTS file. The DOTS message must exactly match the `height` and `width` that you configured.

Display Sequence

The `sequence` element defines the order in which TEXT files are to be displayed. Include a list of TEXT file labels.

```
<alphasign>
  <sequence labels="AbACAdA"/>
</alphasign>
```

You may also specify two optional attributes, `mode` and `locked`. Values for the `mode` attribute are as follows.

- `useTimeSchedule` - The specified TEXT files will always run according to the configured time schedule. This is the default value.
- `ignoreTimeSchedule` - The specified TEXT files will always run, regardless of each file's time schedule.
- `deleteAtStopTime` - The specified TEXT files will run according to their time schedule, and they will be deleted when the scheduled stop time is reached.

If `mode` is not specified, then the files will run according to their time schedule.

The `locked` attribute may be set to "true" to prevent the files from being edited with the remote control. The default is "false".

```
<alphasign>
  <sequence labels="AbACAdA" mode="ignoreTimeSchedule" locked="true"/>
</alphasign>
```

Day Schedule Table

Set the `dayScheduleTable` when you want certain TEXT files to be displayed only on certain days.

```
<alphasign>
  <dayScheduleTable>
    <daySchedule label="A" start="never" stop="Monday"/>
    <daySchedule label="b" start="Sunday" stop="Tuesday"/>
    <daySchedule label="C" start="Monday-Friday" stop="Saturday"/>
    <daySchedule label="d" start="always" stop="Monday"/>
  </dayScheduleTable>
</alphasign>
```

start values

- `daily`
- `Sunday`
- `Monday`
- `Tuesday`
- `Wednesday`
- `Thursday`
- `Friday`
- `Saturday`

stop values

- `Sunday`
- `Monday`
- `Tuesday`
- `Wednesday`
- `Thursday`
- `Friday`
- `Saturday`

- `Monday-Friday`
- `weekends`
- `always`
- `never`

When the `start` day is `daily`, `Monday-Friday`, `weekends`, `always`, or `never`, the `stop` day is ignored (but it's still required).

Time Schedule Table

Use `timeScheduleTable` to display certain TEXT files at certain times of the day. The `timeSchedule`'s `start` and `stop` values can be set in 10-minute intervals ("`00:00`", "`00:10`", ..., "`23:40`", "`23:50`"), plus "`all day`", "`never`", and "`always`".

```
<alphasign>
  <timeScheduleTable>
    <timeSchedule label="A" start="always"  stop="11:30"/>
    <timeSchedule label="b" start="all day" stop="18:00"/>
    <timeSchedule label="C" start="never"   stop="00:00"/>
    <timeSchedule label="d" start="08:10"   stop="15:50"/>
  </timeScheduleTable>
</alphasign>
```

When the start time is set to "always", the sign ignores the stop time (but it's still required). I assume that the same is true for "all day" and "never", but the protocol doc doesn't explicity say.

Time

To set the sign's internal clock, use `timeOfDay` with its two required attributes, `hour` and `minute`. Valid values for `hour` are "`00`"-"`23`", and values for minute are "`00`"-"`59`".

```
<alphasign>
  <timeOfDay hour="19" minute="55"/>
</alphasign>
```

Day of Week

If your sign has an internal calendar (BetaBrites don't), you can set the day of week with `dayOfWeek`.

```
<alphasign>
  <dayOfWeek day="Monday"/>
</alphasign>
```

Valid values for the required `day` attribute:

- `Sunday`
- `Monday`
- `Tuesday`
- `Wednesday`
- `Thursday`
- `Friday`
- `Saturday`

Set Speaker Mode

To enable or disable the sign's speaker, use `speakerMode` with the `enabled` attribute set to "`true`" or "`false`".

```
<alphasign>
  <speakerMode enabled="true"/>
</alphasign>
```

Set Time Format

Set the time format using the `timeFormat` element. Set the required `format` attribute to either "`am-pm`" or "`24-hour`".

```
<alphasign>
  <timeFormat format="24-hour"/>
</alphasign>
```

This controls the way the `time` value is displayed.

Soft Reset

Send `softReset` to reboot the sign. This is non-destructive; memory will not be cleared.

```
<alphasign>
  <softReset/>
</alphasign>
```

Beep

To sound the beeper, use the `beep` element.

```
<alphasign>
  <beep/>
</alphasign>
```

The `type` attribute must be set to "`on`", "`off`", "`long`", or "`short`".

```
<alphasign>
  <beep type="long"/>
</alphasign>
```

Please note that "`on`" and "`off`" should not be used with signs that have standard speaker/piezo alarms, as it may cause damage to the sign. There is a warning about this in the Alpha Sign Protocol documentation.

On some signs (not Betabrite), you can use `frequency`, `duration`, and `repeat` attributes instead of the `type` attribute.

```
<alphasign>
  <beep frequency="120" duration="15" repeat="4"/>
</alphasign>
```

Valid values:

- `frequency` - `0-254`
- `duration` - `1-15` *(deciseconds)*
- `repeat` - `0-15`

Set Dimming Mode

If your sign supports it, you can set the dimming mode with the `dimmingMode` element and one of two alternative pairs of attributes.

Form one, for Alpha Solar signs:

```
<alphasign>
  <dimmingMode threshold="19" level="3"/>
</alphasign>
```

Valid attribute values:

- `threshold`: `0-21`, where `0` means no dimming, `1` means darkest outside, and `21` means brightest outside
- `level`: `0-4`, where `0` is 100% brightness and `4` is 44% brightness

Form two, for Big Dot signs:

```
<alphasign>
  <dimmingMode start="20:30" stop="06:00"/>
</alphasign>
```

Valid `start` and `stop` time values are the same as for timeScheduleTable.

Clear Serial Error Status Register

Send `clearSerialErrorStatusRegister` to clear the serial error status register. See the Alpha Sign Protocol documentation for an explanation.

```
<alphasign>
```

```
  <clearSerialErrorStatusRegister/>
</alphasign>
```

Set Calendar Date

Send `calendarDate` to set the calendar date, if your sign supports it.

```
<alphasign>
  <calendarDate month="02" day="04" year="06"/>
</alphasign>
```

Configure Counters

Counters are a feature of Alpha signs. Betabrite signs do not support counters. A counter is a value stored inside the sign that can be incremented automatically once every minute, hour, or day. When a counter reaches its target value, it can trigger the display of a TEXT file. The value of the counter can be displayed in a text message. See counterValue.

The `counterConfig` element allows you to configure the sign's built-in counters. It contains one or more `counter` child elements. Each of these configures a specific counter. Alpha signs have 5 counters, numbered 1-5.

The `counter` element has one attribute, `id`, which identifies the counter. This is a 1-digit number, `1-5`. It also has 3 child elements: `counterOptions`, `counterValues`, and `counterSchedule`, which specify the configuration parameters.

The `counterOptions` element specifies three attributes:

- `enabled`: (`true` | `false`) enabled or disables the counter
- `eventToCount`: (`minutes` | `hours` | `days`) If set to `minutes`, the counter will increment whenever the clock minute changes. If set to `hours`, it will increment whenever the hour changes. If `days`, it will increment at midnight, when the day changes.
- `autoReset`: (`true` | `false`) If true, the counter will reset to its start value after it reaches its target value. If false, it will stay at its target value.

The `counterValues` element specifies four attributes, all integers.

- `start`: (`0 - 99`) The starting value of the counter. If autoReset is enabled, this is the value that the counter will reset to.
- `increment`: (`-99 - 99`) The increment value. Whenever a new `eventToCount` occurs, the counter will be incremented by this amount. This value can be negative, producing a countdown.
- `current`: (`0 - 99`) The current value.
- `target`: (`0 - 99`) The target value. When the counter reaches the target value, it will turn on the TEXT file with a label equal to the counter's id attribute. That is, counter 1 will call text 1, counter 2 will call text 2, etc.

The `counterSchedule` element specifies three attributes.

- `start`: This is a time value which specifies the starting time for the counter. Legal values are the same as for timeScheduleTable.
- `stop`: This is another time value which specifies the stopping time for the counter. Legal values are the same as for timeScheduleTable.
- `weekends`: (`true` | `false`) If true, the counter will continue counting events on Saturday and Sunday. If false, it will only count on Monday-Friday.

```
<alphasign>
  <counterConfig>
    <counter id="1">
      <counterOptions enabled="true" eventToCount="minutes" autoReset="true"/>
      <counterValues start="1" increment="1" current="1" target="4"/>
      <counterSchedule start="00:00" stop="12:00" weekends="true"/>
    </counter>
    <counter id="2">
      <counterOptions enabled="false" eventToCount="days" autoReset="false"/>
      <counterValues start="1" increment="1" current="1" target="5"/>
      <counterSchedule start="00:00" stop="never" weekends="true"/>
    </counter>
  </counterConfig>
</alphasign>
```

Counter Value

The counter value can be retrieved with `counterValue` and its `id` attribute.

```
<alphasign>
  <text label="0">count: <counterValue id="0"/></text>
```

```
</alphasign>
```

Set Temperature Display Units

The `temperatureDisplay` element has one attribute, `units`, which specifies "F" or "C" as the display units.

```
<alphasign>
  <temperatureDisplay units="F"/>
</alphasign>
```

Set Temperature Offset Amount

The `temperatureOffset` element specifies a correction amount to add to the reading of the sign's internal thermometer. For all Alpha signs except Solar, the amount is sent in the `offset` attribute. This value is a single digit integer, positive or negative, `-9` - `9`. For Solar signs, send the actual temperature in the `temperature` attribute, and the sign will compute the offset. (It is unclear from the documentation what units to use; unless someone tells me otherwise, I assume that it is in the units specifed by temperatureDisplay configuration.)

```
<alphasign>
  <temperatureOffset offset="-5"/>
</alphasign>
```

For Alpha Solar signs:

```
<alphasign>
  <temperatureOffset temperature="73"/>
</alphasign>
```

Set Serial Address

The `signAddress` element changes the sign's serial address. This is the same address that you must use in the alphasign element's `signAddress` attribute. It is a 2 character uppercase hex value, 00 - FF.

```
<alphasign>
  <signAddress address="01"/>
</alphasign>
```

Display Text at XY Position

The `xyTextMode` element enables or disables the XY positioning mode. The child element, `xyText`, with its two required attributes, `x` and `y`, set text at an XY position. To use XY positioning, you must first enable it with an empty `xyTextMode` element. Then, you can send one or more `xyTextMode` commands with `xyText` child elements. Finally, you must turn off XY positioning by sending `xyTextMode` again, with the `enabled="false"` attribute.

While in XY mode, the sign will ignore all other control codes, except for color selection and the 5-high or 7-high character codes.

```
<alphasign>
  <!-- first, enable XY mode -->
  <xyTextMode/>
  <!-- next, send XY text -->
  <xyTextMode>
    <xyText x="1" y="2">Hello world!</xyText>
    <xyText x="2" y="4"><standard5/>What's <standard7/>up<standard5/>, doc?</xyText>
    <xyText x="3" y="10"><colormix/>Goodbye <red/>world!</xyText>
  </xyTextMode>
  <!-- finally, disable XY mode -->
  <xyTextMode enabled="false"/>
</alphasign>
```

Read Commands

Although many of the "read" commands are included, BBXML has no provision for actually reading the data from the serial port or converting it into XML. These commands only send the instruction that makes the sign return the data. Consult the Alpha Sign Protocol documentation for information about the message format of the returned data.

The following "read" commands are supported.

- `readText label="`*`label`*`"`
- `readString label="`*`label`*`"`

- `readDots label="label"`
- `readLargeDots label="label"`
- `readRGBDots label="label"`
- `readTimeOfDay`
- `readSpeakerMode`
- `readGeneralInfo`
- `readMemoryPoolSize`
- `readMemoryConfig`
- `readMemoryDump`
- `readDayOfWeek`
- `readTimeFormat`
- `readTimeScheduleTable`
- `readSerialErrorStatusRegister`
- `readNetworkQuery`
- `readSequence`
- `readDayScheduleTable`
- `readCounters`
- `readAlphavisionDOTSMemoryConfig`
- `readRunFileTimes`
- `readDate`
- `readDaylightSavingTime`
- `readAutoModeTable`
- `readTemperatureOffset`

```
<alphasign>
  <readText label="B"/>
</alphasign>
```

```
<alphasign>
  <readString label="s"/>
</alphasign>
```

```
<alphasign>
  <readTimeOfDay/>
</alphasign>
```

Sending Several Commands at Once

You can combine several commands into a single message, as long as you keep them in a sequence that the sign can handle. The messages are processed by the sign in the order that they are received -- that is, in the order that you write them. Certain combinations don't work together. In particular, you have to send the `beep` command last, because the sign's serial port is disabled while its beeper is beeping.

```
<alphasign>
  <string label="d">Sunday, August 1</string>
  <text label="A"><mode display="hold"/><rainbow1/>BetaBrite<red/>!!!</text>
  <text label="D"><mode display="compressedRotate"/><callString label="d"/></text>
  <text label="T"><mode display="wipeUp"/><orange/><cs7HighFancy/><time/></text>
  <timeScheduleTable>
    <timeSchedule label="A" start="Always" stop="00:00"/>
    <timeSchedule label="T" start="Always" stop="00:00"/>
    <timeSchedule label="D" start="06:00"  stop="11:00"/>
    <timeSchedule label="m" start="Always" stop="00:00"/>
  </timeScheduleTable>
  <sequence labels="ADTD"/>
  <speakerMode enabled="true"/>
  <beep type="short"/>
</alphasign>
```

There is one more restriction to keep in mind. The size of the entire message must be less than the amount of free memory in the sign. If you do not have much free memory left, you may have to send each command separately. This is especially true if you are sending several big DOTS files.

Example Application

Once you learn to use XML and XSLT, it becomes very easy to put information on your sign. Consider an application that reads CallerID data from a modem and outputs it as XML. We can use XSLT to transform the CallerID XML into a BBXML message.

Here is our `callerID.xml` file:

```
<callerID>
  <date>0429</date>
  <time>0050</time>
  <nmbr>8885551212</nmbr>
  <name>FRED BLOGGS</name>
  <mesg/>
</callerID>
```

Here is the `bbCallerID.xsl` file:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.1">
  <xsl:output indent="yes"/>
  <xsl:template match="text()|@*"/> <!-- suppress default output -->

  <xsl:template match="callerID">
    <xsl:element name="alphasign">
      <xsl:element name="text">
        <xsl:attribute name="label">0</xsl:attribute>
        <xsl:attribute name="mode">rollUp</xsl:attribute>
        <xsl:element name="green"/>
        <xsl:apply-templates select="name"/>
        <xsl:element name="CR"/>
        <xsl:apply-templates select="nmbr"/>
      </xsl:element>
    </xsl:element>
  </xsl:template>

  <xsl:template match="callerID/nmbr">
    <xsl:value-of select="."/>
  </xsl:template>

  <xsl:template match="callerID/name">
    <xsl:value-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

Running that through `xsltproc`, we get this output.

```
xsltproc bbCallerId.xsl callerID.xml
```

```
<alphasign>
  <text label="0"><mode display="rollUp"/><green/>FRED BLOGGS<CR/>8885551212</text>
</alphasign>
```

You can use the XSLT `substring()` function to format the phone number with hyphens. You might also want to handle the special nmbr values, 'P' and 'O', to print "Private" and "Out of Area", respectively. This is left as an exercise for the reader.

Send it straight to the sign like this:

```
xsltproc bbCallerId.xsl callerID.xml | bbxml
```

Unsupported Features

"LARGE DOTS PICTURE" file, a feature of certain large Alpha signs, is not supported.

Although many of the advanced Alpha sign features are included, I have not tested this with anything but my Betabrite 1040 and Alpha 215 signs. Let me know if you have any problems.

License

[up]
Darin Franklin