# WBTV 1 : Publish-Subscribe for Embedded Devices

## Introduction

This page presents a simple protocol for realizing publish-subscribe messaging in small low cost embedded devices using ordinary UART(8-n-1) over any wired-AND physical layer.

The protocol can be implemented in about 3K of memory+ 70 bytes of RAM.

The protocol has no special dependence on timing of byte arrivals and does not rely on timeouts to detect a node that has failed during transmission.

## Semantics

Each message can be thought of as pushing a “message” made of “segments” to a “channel”. Both the message segments and the name of the channel are arbitrary 8 bit strings.

All channels with <=6 byte names are reserved for future standardization.

To avoid conflicts, channel names should be based on URLs or UUIDs or some other unique string, but can simply be chosen arbitrarily when global compatibility is not required.

Channels can be thought of as inputs or outputs or both of the node that “owns” the channel.

Devices can use a common prefix for all their owned channels but are not required to.

## Message Structure

Every message begins with an exclamation point, followed by the N byte channel name, then a tilde as a separator , then the data, then a 2 byte

checksum, then a closing newline(ASCII 10). All data should always be little-endian.

Should the message contain multiple segments, separate them with un-escaped tildes.

## Escaping

Should a control symbol(!, ~, or \n) need to appear in data or channel name, the backslash is used to escape it. The character following a non-escaped backslash(even another backslash) is processed as a literal character.

## Checksum

The checksum is a fletcher-256 sum, as defined by the following function :

```
set fast and slow to zero
then for each byte to be hashed:
     increment slow by the byte mod 256
     increment fast by slow mod 256
```

Calculated over all the bytes of the raw channel, the tilde, and the raw message, but not including the escapes or any other control characters except for the tilde. The checksum is appended to the message slow byte first, making the full message structure(control chars underlined):

```
!<channel>~<message><fast><slow>\n
```

## CSMA

To remove the need for a bus master, we use the wired-OR Physical Layer and CSMA/CD. Before sending a message, a device must pick a random length of time, and wait until the bus has been free for that time.

Devices must also listen while transmitting, and, if they detect a collision(via the wired-or), immediately cease, wait another random period(preferably 2x the initial period) and retry.

# WBTV 2: Time Synchronization Protocol

## Introduction

Often there is a need to provide accurate time synchronization to many devices. This document defines a reserved channel for that purpose.

## The TIME Channel

The TIME channel is reserved for broadcasting messages that represent the current time, as of the leading edge of the start bit of the initial ! In the message. Messages must be timestamps conforming to the following format.

## Time Stamp Format

A time stamp contains the following fields, concatenated together, plus any optional fields which may be added, which may be specified in a later document.

### UNIX Time Stamp

This must be a 64-bit signed integer representing the number of seconds elapsed since 00:00 1 January 1970, excluding leap seconds. This is identical to the 64 bit UNIX time number.

This gives enough precision for all of recorded history until millions of years from now.

### Fractional Part

This is a 32 bit unsigned integer representing the fractional part of the current time multiplied by 2**32. Where the fractional part is unknown, 2**16 should be sent to minimize average error.

### Accuracy Estimate

This is an estimate of the accuracy, given as a 8 bit signed integer E followed by am 8 bit unsigned integer M, where the maximum error in seconds is M*(2^E).

This is an extremely simple floating point format that can only represent positive integers.

The estimate should be conservative, and should be the maximum error the sender is rated to experience across the full range of working conditions.

## Device Behavior

Devices should avoid sending time messages during leap seconds, as the time number is ambiguous. Devices should be able to be configured to send time data rather frequently, on the order of up to every several seconds, to account for devices without accurate clocks.

Devices should maintain an estimate of error for their internal clock, and should ignore messages that advertise more error than that, i.e. devices with an accurate estimate of the time should ignore broadcasts from devices with a less accurate estimate of the time.

Devices should not ideally send the time if a time message with equal or better accuracy has been recently sent, specifically, if a receiver with a 2% accurate clock would not have accumulated more error than the internal estimate, there is likely no use in sending the time.

# WBTV 3: Device Discovery and ID

## Introduction

Often it becomes necessary to uniquely identify a device, and to enumerate devices on the bus. This document establishes rules for device IDs for WBTV devices.

## The Device UUID

Every device should, but is not required to, have a 16 byte UUID, which may conform to an official UUID standard of may simply be a random string. A devices UUID should never change, and no two devices should ever have the same UUID. Big endian encoding should be used for all UUIDs used with the WBTV protocol, even though WBTV is normally little endian.

## Friendly Name

Devices should, but are not required to, be able to accept a user-assigned friendly name. The method of assigning a name should be to send a single packet to channel NAME containing the target device ID followed by the name, then the end of packet.

Names may not be longer than 24 characters, and must be valid UTF-8.

Devices should announce their names at power-up or on connection, by sending their device UUID followed by their name to the MYNAME channel.

## The STATUS channel

The channel STATUS is reserved for devices to post short status updates. Messages must be composed of the device's UUID, then optionally a newline and a set of space separated keywords, then by a newline and free text.

Device specific Keywords should begin with “x-”. General Keywords:

| | |
|---|---|
| power-warn | Low input voltage |
| power-fail | Insufficient input power |
| temp-warn | Overtemperature warning |
| temp-fault | Overtemperature fault |
| volt-warn | Overvoltage warning |
| normal | Normal Operation |

Most devices should not send STATUS messages, however for some devices a heartbeat/still connected signal is important.

## The INFO channel

This channel is used by devices to transmit information about themselves to the network.

Messages must be composed of the device's UUID followed by a newline, followed by the manufacturer name, another newline, the product name, another newline, then two newlines and any amount of free description.

## The SCAN channel

Posting a message to the SCAN channel allows one to scan the network for devices. All devices receiving a blank message on this channel should send any information they may have about themselves and channels they own to the network.

Devices should NOT immediately send their info, but should wait a random time from 0 to 20s or so, during which time they should continue normal operation. Devices should only use SCAN with full knowledge that the network may be heavily trafficked from responses.

# WBTV 4: Service Discovery

## Introduction

Automatic network configuration is a common requirement. Consider a wind speed sensor that sends data on a specific channel, which obviously must be different between devices to allow for multiple devices to be used. How can we know what channel a device sends on?

This document defines the concept of a WBTV Service. A WBTV Service is simply a type of service a device can provide on a channel. The wind speed sensor could be thought of as providing a "wind speed service" on the channel it sends on, and a light controller could be thought of as providing a "Light Control Service" on the channel that it listens on.

Services are identified using UUIDs, and as such anyone can define a service, and publish it formally or informally in the appropriate manner, such as online or in a user manual.

Devices that provide services must have a unique UUID that is not shared even between identical devices. This UUID may also be used as a channel name without conflict to save ram.

There is no restriction against two compatible services sharing a channel.

## The SERV Channel

The SERV channel is reserved for service announcements, which should be made after a device powers up, or in response to SCAN messages. and messages must obey the following format:

### Owning Device

The first 16 bytes of a SERV message must contain the UUID of the owning device which is providing the advertised service.

### Service UUID

The next 16 Bytes must be the UUID of the service being provided.

### Channel Len

The length of the channel name of the channel on which thee service is provided in bytes, represented as one byte.

### Channel

The channel name on which the service is provided.

### Modifier Bytes

Any further data is to be interpreted in a service- and device specific manner.

## The Null Service

This is a service with a UUID of 16 zeros. It has no specific definition. One must use the modifier bytes and device documentation to explain the channel.

## The Struct Service

This service is used for simple sensors and the like. The modifier bytes must be C source code containing either or both of two struct definitions, one called input and the other output. Data sent *to* the device must be an unpadded packed little endian instance of *input*. And data sent from the device must match output.

# WBTV 5: The DAQ Service

## WBTV-DAQ Service

This is a WBTV4 Service that allows a device to act as a data acquisition device.

Service UUID(hex):
0x8A9E60340D8611E4843156F31D5D46B0

## Modifier Bytes

The modifier bytes are to be used to define a set of signals, wherein each line defines a signal(the first line defining signal 0) and so on.

Signals 128+ are reserved, with signal 128 representing a UNIX timestamp, consisting of a 16 bit seconds part and 16 bit fractional part,

This will roll over every 2^16 seconds, which is not an issue because arrival time can be used to disambiguate.

Each line is to consist of, at minimum, a type and a name(which may not include spaces), as in the example:

```
u16 Position
```

Which would represent a signal called position, which is represented as a 16 bit unsigned integer.

### Valid Types

The types that are defined in this document are:

#### u8, u16, u32, u64, u128

Unsigned integer of specified bit width

#### i8, i16, i32, i64, i128

Signed integer of specified bit width.

#### f16, f32, f64, f128

IEEE Float of specified bit width.

### Keywords

The name and type may optionally be followed by any number of keywords, some of which may have arguments, That may appear in any order.

#### in <unit>

Specifies a unit or interpretation to be used with the channel. <unit> must not contain any spaces, and should where possible be an SI base or derived unit, with ^ used to denote exponentiation where needed e.g. for "mW/cm^2" Example:

```
u16 speed in mm/s
```

#### range <min>:<max>

Applies to numeric types. Specifies that the number reported for this signal will be in the specified range. Expressions may be used containing ^*/+- as needed. No whitespace may appear in numeric expressions, now between the numeric expression and the colon.

Si prefixes should be preferred over expressions(mV instead of V/1000)

To represent ratios, a number can be a unit, e.g. "u8 filllevel in 255", denotes that 255 is full and 0 is empty.

## Message Format

Messages sent by the device on the service channel must consist of a set of *datums*, wherein each datum is a data point from a certain signal, and is preceded by it's signal number.

For example. A DAQ service channel with the modifier bytes:

```
u16 Voltage in mV
```

Might send a message consisting of a zero, indicating signal 0, followed by the two bytes of the 16 bit unsigned int, followed by the signal number and data for signal 0(if one existed), etc.

Devices may send one or many datum points in this way, and are not required to send all points.

# WBTV 6: DAQ Service Extensions

## Introduction

WBTV5 Defines a service allowing devices to act as data acquisition units. This document defines extensions to the DAQ service as defined in WBTV5.

## Bit Arrays

A bit array is a field with type bits8, bits16,bits32, or bits64.

They are logically the same as unsigned integers, but are intended to be interpreted as packed boolean arrays.

Should the "in" keyword be used to give an interpretation to the data, the interpretation should consist of a comma separated list, containing no whitespace, giving a name to each bit in the bit array, with the first name applying to the least significant bit. Example:

```
bits8 buttons in A,B,X,Y,L,R,S,s
```

Represents one byte, representing the values of 8 buttons, as one might find on a game controller.

When the device sends it's values, a 1 in any bit position should be used to represent True, and a 0 should be used to represent False.

## The Output Control Service

The output control service is a service used to allow a device to act as an output in the same way as the DAQ service allows devices to act as inputs.

UUID(Hex):
0xCAEDB14E0D8611E4B78578F31D5D46B0

The output control service has identical semantics to the input service, except the device listens instead of transmits on the channel.

The modifier bits are used as in the DAQ service to define a set of signals in an identical manner.

For example:

```
u8 brightness
```

When found in the modifier bits of a Output Control Service Channel would mean the device was listening for a message on that channel containing a 8 bit value in signal 0, which the device would use to set the brightness of some light, presumably.

A message sent on that channel might contain:

```
0x00, 0xff
```

To tell the device to go to full brightness.

Or:

```
0x00, 0x00
```

To turn the light off.

# WBTV 7: Stage Light Control

## Introduction

The WBTV Stage light control protocol provides a standard way to control stage lighting appliances, and a way for them to provide information about themselves using a WBTV4 Service Channel.

The general model is an array of 2**16 8 bit values, which may be set, or smoothly faded.

Each device listens to a range of values determined by it's "start code", the first value in it's range of addresses to listen to, set in a device-specific manner such as via a special channel or with DIP switches.

Each device only needs to store info about the channels it listens to, but for efficiency, one may write to the imaginary shared space of addresses as if a single device were listening.

## The STAGE Channel

This channel is used to set and fade lighting values. Messages are a stream of commands, where each command has a command byte followed by any arguments. Devices should halt and discard the remainder upon encountering an unknown command.

The total length of a STAGE message should be under 128 bytes.

Compatible devices receiving a STAGE message should execute the commands sequentially, and ignore all values written or faded that do not overlap their range.

Devices should internally use as many bits of precision as possible for the smoothest fading.

## Commands

### 0x00 Set Values

This command directly writes to the virtual 2**16 byte space. Structure(first part at the top):

| | |
|---|---|
| Fixed uint8 0x00 | Command Code |
| Uint16 spos | Start Position |
| Uint8 valcount | Number of values to write |
| uint8*N data | Values to be written |

Where the length of data is valcount. Any fades on any values not referenced in the command should be unaffected.

### 0x01 Fade Values

This command begins a fade from the current values to the new values over a period of time. Structure:

| | |
|---|---|
| Fixed uint8 0x01 | command code |
| uint16 spos | Start Address |
| uint8 valcount | Number of values to write |
| uint8 duration | Fade duration in 48ths of a second |
| uint8*N data | Fade destination values |

Causes *valcount* values starting at *spos* to fade from their current values to the destination values specified in data.

Any fades on any values not included in the command should continue unaffected.

Where the concept of a fade is nonsensical, such as on-off only lights, fades may be ignored.

# WBTV 8: Stage Light Control Extensions

## Introduction

The WBTV STAGE protocol defines a way of controlling lighting fixtures. This document defines an extension to the STAGE protocol allowing higher resolution control.

## Labeling

Devices that are compatible with the STAGE protocol as laid out in WBTV 7 should be labeled as supporting "WBTV STAGE". Devices also supporting all features of this document should be labeled as supporting "WBTV STAGE 2". STAGE 2 devices must also support all commands of WBTV 7.

## 16 bit commands

16 bit STAGE commands use the same value space array as 8 bit commands. A write or fade to channel 5 should do exactly the same thing in 16 or 8 bit mode, besides the higher resolution.

Where possible, fades between 8 bit values should be handled internally as 16 bits for maximum smoothness.

## Commands

### 0x02 16 bit Set Values

This command directly writes to the virtual 2**16 byte space. Structure(first part at the top):

| | |
|---|---|
| Fixed uint8 0x00 | Command Code |
| Uint16 spos | Start Position |
| Uint8 valcount | Number of values to write |
| Uint16 *N data | Values to be written |

Where the length of data is valcount. Any fades on any values not referenced in the command should be unaffected.

### 0x03 16 bit Fade Values

This command begins a fade from the current values to the new values over a period of time. Structure:

| | |
|---|---|
| Fixed uint8 0x01 | command code |
| uint16 spos | Start Address |
| uint8 valcount | Number of values to write |
| uint8 duration | Fade duration in 48ths of a second |
| uint16*N data | Fade destination values |

Causes *valcount* values starting at *spos* to fade from their current values to the destination values specified in data.
Any fades on any values not included in the command should continue unaffected.
Where the concept of a fade is nonsensical, such as on-off only lights, fades may be ignored.

# WBTV 9: Misc

## Introduction

This Document defines assorted things.

## The RAND channel

The RAND channel is to be used for random numbers. Hardware RNG devices may send random bytes on this channel, but only high quality, whitened bytes such as produced by radioactive decay or diode noise. Messages can contain 1-64 random bytes. This channel should only be used for “ready to use” bytes.

One should obviously not use these bytes for anything important if there is a possibility of rouge devices connected to the network that might spy on bytes.

## The TZOS Channel

This channel allows a device to broadcast the time offset from UTC of any time zone. Messages should consist of the time zone name(as in “US/Pacific”, followed by a null and then the current offset from UTC(The number that, when added to UTC, would produce the time in the time zone), as a 32 bit integer.
This may optionally be followed by a 32 bit unsigned integer containing the UNIX timestamp of a future time, followed by the time offset at that time as a 32 bit unsigned integer.

## The TZRQ channel

Allows a device to request the time in a specified time zone from any device that knows it. The message is simply the name of the time zone.

A device that knows the answer may respond with a TZOS message.

## The LCTZ channel

Messages on this channel should be the name of the local time zone of the device that is sending the message, optionally followed by a NUL and then the 32 bit integer offset from UTC in seconds of that zone.

## The Struct In Service:

This service allows a device to recieve data formatted as an unpadded C struct. The modifier bytes must be a valid C struct definition, which should contain comments. The struct may be named anything and should use standard typedefs(uint8_t,etc). Data sent to the device must be formatted as per the struct.
UUID:
0xDD452C900D8711E4813782F41D5D46B0

## The Struct Out Service:

Same as the struct in service, except the device sends instead of receives the struct-formatted messages.
UUID:
0xED5690880D8711E4BA3289F41D5D46B0

## The NOTE channel

This channel is used for a device to send freeform UTF8 text that is not intended to be read by a machine, such as sending copyright info or a manufacturer URL on bootup.

## The EVENT channel

This channel is used for announcing or causing events to occur. Messages must be 16 byte UUIDs representing the event.

# WBTV 10: Physical Layer

## Introduction

This document defines several physical layer variants for WBTV.

## WBTV-A

8-N-1 Serial carried over CAN signaling levels, where a logical 1 is represented by the idle or recessive state, carried over CAT5 cable using normal RJ45 jacks.

| **RJ45 Pin** | **Function** |
|---|---|
| 1 | CAN_H |
| 2 | CAN_L |
| 3 | Reserved |
| 4 | Reserved/POWER |
| 5 | Reserved/POWER |
| 6 | Reserved |
| 7 | GND |
| 8 | GND |

Both grounds should be connected together. This provides protection against large ground offsets destroying devices.

If power is supplied, the total current on BOTH LINES combined is not to exceed 600ma, the voltage is not to exceed 32V with brief 35v transients allowed,

Devices must be fully hot pluggable, and must survive a hot plug to 32V via any amount of cable.

Devices with a "pass-through" port should pass through all pins directly unless a specific reason exists to do otherwise.

Devices must function properly with + or − 5 volts of offset between the grounds of two devices

Any devices making use of the reserved pins for other purpose should be clearly labeled with what pins they use. Besides the use of 4 and 5 for power all use of reserved pins should be considered a site specific use.

Devices(Including side-specific uses of reserved pins) must sustain no damage from any combination of prolonged shorts between any pins, even with 35V power applied. Devices must survive -24V applied to the CAN inputs.

Any device capable of supplying current must be current limited to 600ma to prevent fire.

Devices requiring appreciable power should not draw current from the bus and should instead use an external power supply. Devices making use of an external power supply must connect their local ground to bus ground via a 3300 or 4700 ohm resistor with a power rating of at least one half watt.

Devices must keep input current ripple to a minimum, especially in the audio band.

## WBTV-B

One wire with a pullup resistor of 3.3k to 100k to 3.3v, idle high, non inverted, 8-N-1, used as a wired-or bus in the manner of I2C. S

## WBTB-USB

Full-duplex(No CSMA) over a virtual USB COM port.

## WBTV-U

Full duplex serial over standard TTL level signalling. Should be written with the voltage used(e.g. WBTV-U(5V)).