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:
!<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.