gobus protocol

version 1.50 (beta)

14 September 2012
Secret Labs LLC
www.netduino.com

highlights

- Virtual I/O profiles auto-enumerate and add features (peripherals) of slave MCUs to master MCU
- Up to 128 instances of each I/O profile per device (UART, GPIO, ADC, PWM, STREAM, etc.)
- 255 GoBus devices supported per bus
- Up to 63 concurrent function calls (threads) per device
- No data stream interruptions:
  Up to 15 simultaneous outstanding frames per device, ACKs included with every frame
- Media independent, supports multiple transport options:
  Synchronous fixed-length frames (e.g. SPI frame exchange)
  Asynchronous variable-length frames (e.g. UART)

gobus frame format

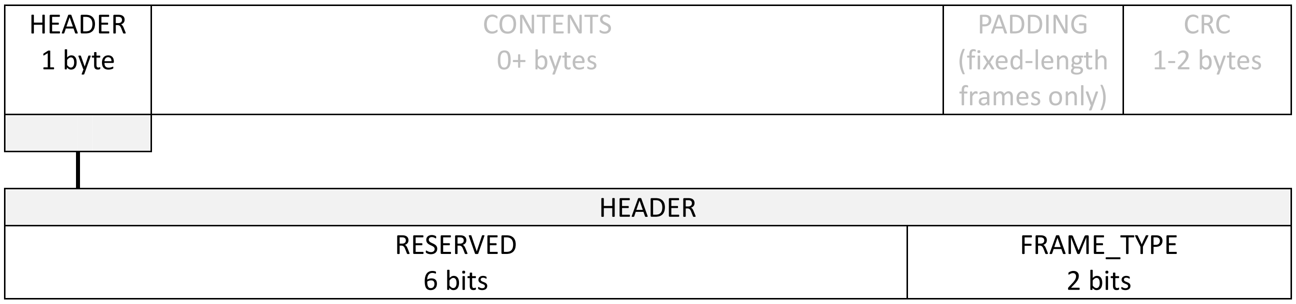
<u>GOBUS FRAME FORMAT (all frame types)</u>

All GoBus frames are constructed with a HEADER byte, optional PADDING bytes, and a trailing CRC.

General format for ALL frames:

HEADER 1 byte	CONTENTS 0+ bytes	PADDING (fixed-length frames only)	CRC 1-2 bytes

HEADER is an 8-bit value which includes 6 reserved bits and a 2-bit FRAME_TYPE value.



HEADER > RESERVED should be populated with zeros and not be interpreted (except in RESET frames).

HEADER > FRAME_TYPE has four options (mask: 0x03)

- 0x0 = EMPTY frame (fixed-length frames only)
- 0x1 = DATA frame
- 0x2 = RESET frame
- 0x3 = NAK frame

CONTENTS contains the frame's data. Interpretation of CONTENTS is specific to the frame type.

PADDING is all-zero data used to pad fixed-length frames. PADDING is used with synchronous frame-exchange transport types such as SPI. Fixed-length frames enable high-speed communication including DMA for SPI frame exchange and CRC calculation.

CRC is an 8-bit or 16-bit CRC value, as selected by the current frame transport configuration.

At reset, the frame transport uses CRC-8-CCITT.

CRC details (reciprocal and polynomial):

CRC8 (CRC-8-CCITT) -- supports frame length of 18-63 bytes
0x07 = $x^8 + x^2 + x + 1$

CRC16 (CRC-16-CCITT) -- supports frame length of 20-16,383 bytes
0x1021 = $x^{16} + x^{12} + x^{5} + 1$

Format for EMPTY frames: (fixed-length frames only)

00000000 HEADER 1 byte	0000…0000 PADDING 0+ bytes	0000…0000 CRC 1-2 bytes

HEADER > FRAME_TYPE is 0x0.

PADDING is always used with EMPTY frames, since EMPTY frames are used exclusively with fixed-length frames (i.e. synchronous frame exchange protocols such as SPI).

PADDING should be filled with all zero bits.

There are no frame **CONTENTS** because the frame is empty. CONTENTS is omitted from the diagram (since it is empty and 0 bytes in length).

CRC is an 8-bit or 16-bit CRC value, as selected by the current frame transport configuration. This value will always be 0x00 or 0x0000 for EMPTY frames.

In effect, the entire EMPTY frame is all zeros. During frame exchange of fixed-length frames, a transmission of all zeros indicates that no data or acknowledgements are pending.

Format for RESET frames:

#0000010 HEADER 1 byte	RESET_REQUEST (16 x 0xFF) or 128-bit RESET_UUID RESPONSE 16 bytes	CRC8 1 byte

HEADER		
FULL_RESET 1 bit	00000 RESERVED 5 bits	10 FRAME_TYPE 2 bits

HEADER > FRAME_TYPE is 0x2.

HEADER > FULL_RESET bit is set to 1 to request that the slave reboot or to signal that the slave has just booted.

CONTENTS of RESET frames always contains one of two 128-bit (16-byte) values:

- RESET_REQUEST
  {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}
- RESET_UUID (which indicates that the slave device or transport has been reset)
  {unique 128-bit UUID which is not all 0xFF bytes}

If RESET_REQUEST is received by a slave, it reads the FULL_RESET bit. If FULL_RESET is set to 1, then the slave will reboot. If FULL_RESET is 0, then the slave does not reboot but immediately resets the following transport settings to their defaults:

- Frame length: 18 bytes
- CRC: CRC-8-CCITT
- Transmit window size: single-frame (size = 1)
- FRAME_SEQUENCE_NUMBER reset to 0; ACK_SEQUENCE_NUMBER reset to 15

When the transport is reset to its default settings (and any time that the transport settings change), any previously-queued frames must be repackaged with the new transport settings and reset sequence numbers. The master will reset its transport settings to match before sending the RESET_REQUEST.

Slave devices are not required to support non-default transport settings. The settings options are designed to increase throughput and enhance features, which is not a concern for many simple implementations.

At power-up and after device reset, a slave sends a RESET_UUID message with the FULL_RESET bit set to 1. This enables hot-plug in very specific scenarios and notifies the master that the slave is in its default state (power-up). If using SPI transport, the INT pin should then be asserted to indicate that data is waiting to be transmitted.

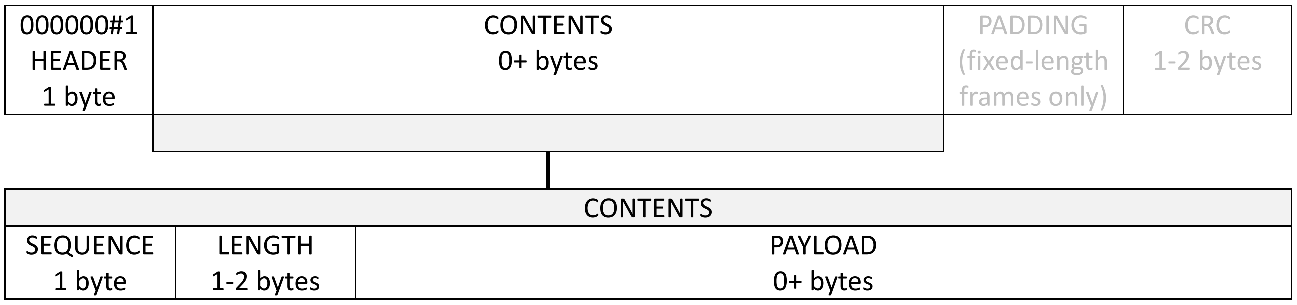
If a slave is currently being used and a RESET_UUID message is received with its FULL_RESET bit set to 1, the master should invalidate or dispose of the slave and its peripherals.   Rebooting while holding an active GoBus connection is not permitted and would result in an invalid state.

There is no frame **PADDING** because RESET frames are always exactly 18 bytes in length.

Devices using fixed-length frames greater than 18 bytes in length must check ‘HEADER > FRAME_TYPE’ to watch for RESET frames before evaluating CRC or disposing of the frame.

Processing of RESET_REQUEST always takes priority over all other transport traffic.

Format for DATA and NAK frames:

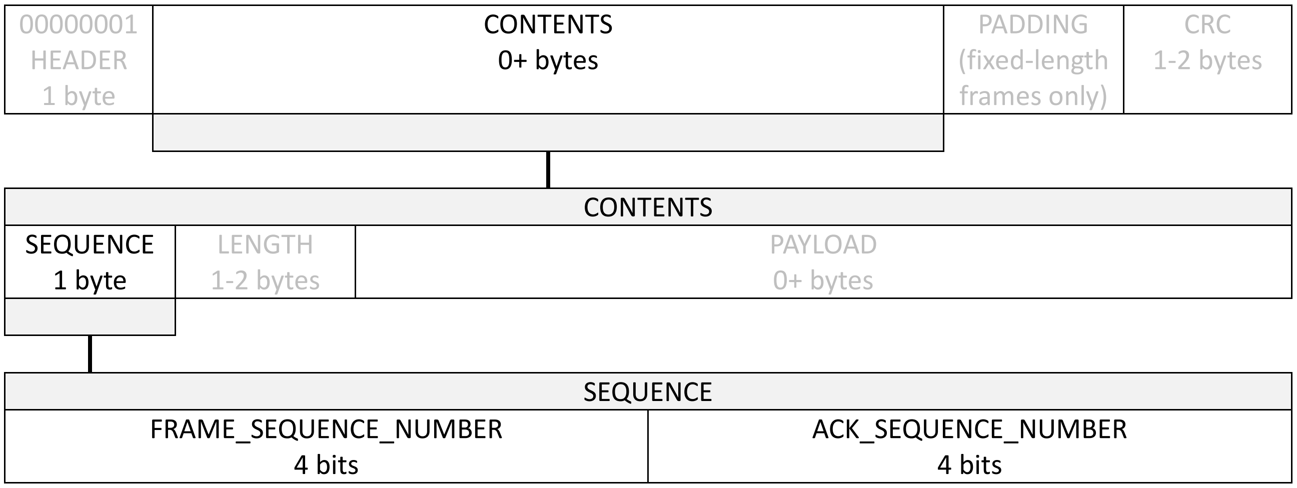


HEADER > FRAME_TYPE is 0x1 for DATA frames. **HEADER > FRAME TYPE** is 0x3 for NAK frames.

NAK frames are DATA frames which carry an indicator that frame corruption has occurred (due to either CRC validation failure or out-of-sequence frames). Both frames should otherwise be handled identically.

CONTENTS of DATA frames contains SEQUENCE numbers, payload LENGTH, and the data PAYLOAD.

CONTENTS > SEQUENCE contains sequence numbers for the outgoing frame and the last frame which was "successfully received."



CONTENTS > SEQUENCE > FRAME_SEQUENCE_NUMBER is the sequence number of the outgoing frame. At transport reset, this value is 0x0. This number is incremented sequentially for each new frame (rolling over from 0xF to 0x0). Up to 15 frames can be sent before being acknowledged, depending on the current transport settings.

If the maximum number of frames have been sent and no ACKs have been received, the transmitter must pause or retry communication.

If a FRAME_SEQUENCE_NUMBER is received out of order or a frame fails CRC verification, then the receiver must ignore the frame and transmit a NAK frame indicating the last successfully ACK'd frame sequence number.

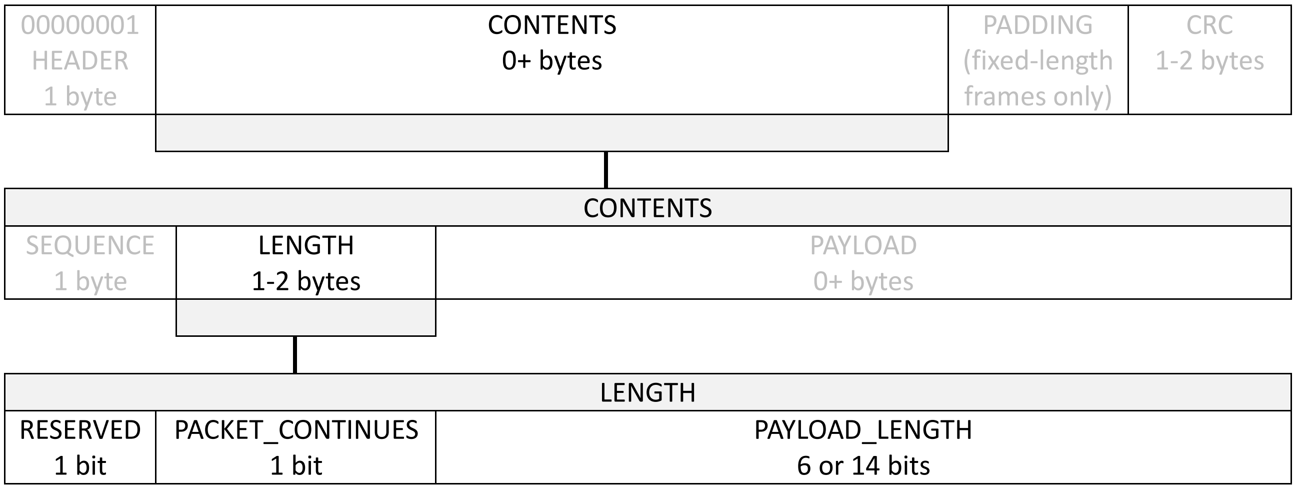
If the incoming frame has an empty PAYLOAD (i.e. the frame is only an "ACK/NAK" and has no payload) then the receiver should not read the FRAME_SEQUENCE_NUMBER and should not send an ACK in response.

If an "ACK-only frame" is corrupted and the frames are subsequently retransmitted, the receiver will send a NAK to indicate that the redundant frames are not processed—but that all frames up to ACK_SEQUENCE_NUMBER have been received.

CONTENTS > SEQUENCE > ACK_SEQUENCE_NUMBER is the sequence number of the most recent "successfully received" frame. Since multiple frames may be received before a new frame is sent, this number may skip sequence numbers to reflect the most recent "successfully received" frame.

If the incoming frame is a NAK frame, the receiver should retransmit all outstanding frames which have sequence numbers at least one greater than "CONTENTS > SEQUENCE > ACK_SEQUENCE_NUMBER".

CONTENTS > LENGTH contains the PAYLOAD_LENGTH and a bit to support multiple-frame packets.



CONTENTS > LENGTH > RESERVED should be populated with zero and should not be interpreted.

CONTENTS > LENGTH > PACKET_CONTINUES indicates that a frame is part of a larger packet.

If a device does not support multi-frame packets, then the PACKET_CONTINUES bit does not need to be processed; every frame will therefore be a single-frame packet and can be processed immediately.

If a device does support multi-frame packets, incoming frames will be appended to an incoming packet buffer. If PACKET_CONTINUES is set to 1, the current frame is either the first frame or a middle frame in a multi-frame packet. If PACKET_CONTINUES is set to 0, the current frame is either a single-frame packet or the last frame in a multi-frame packet.

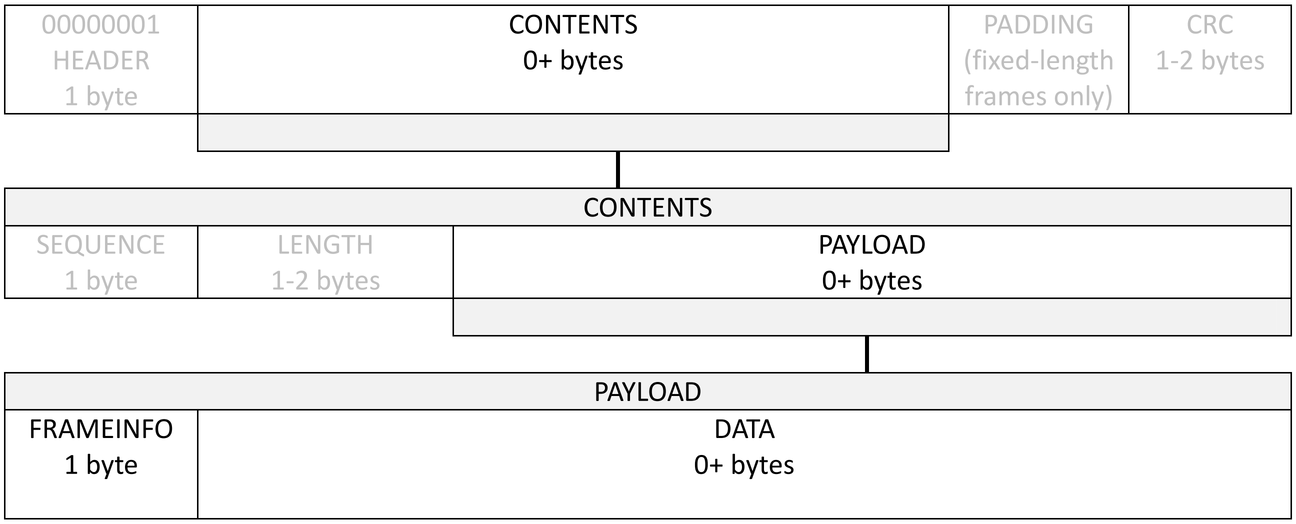
Multi-frame packet support is an optional feature for GoBus-enabled devices and is disabled by default.

CONTENTS > LENGTH > PAYLOAD_LENGTH indicates the length of the data frame's PAYLOAD.

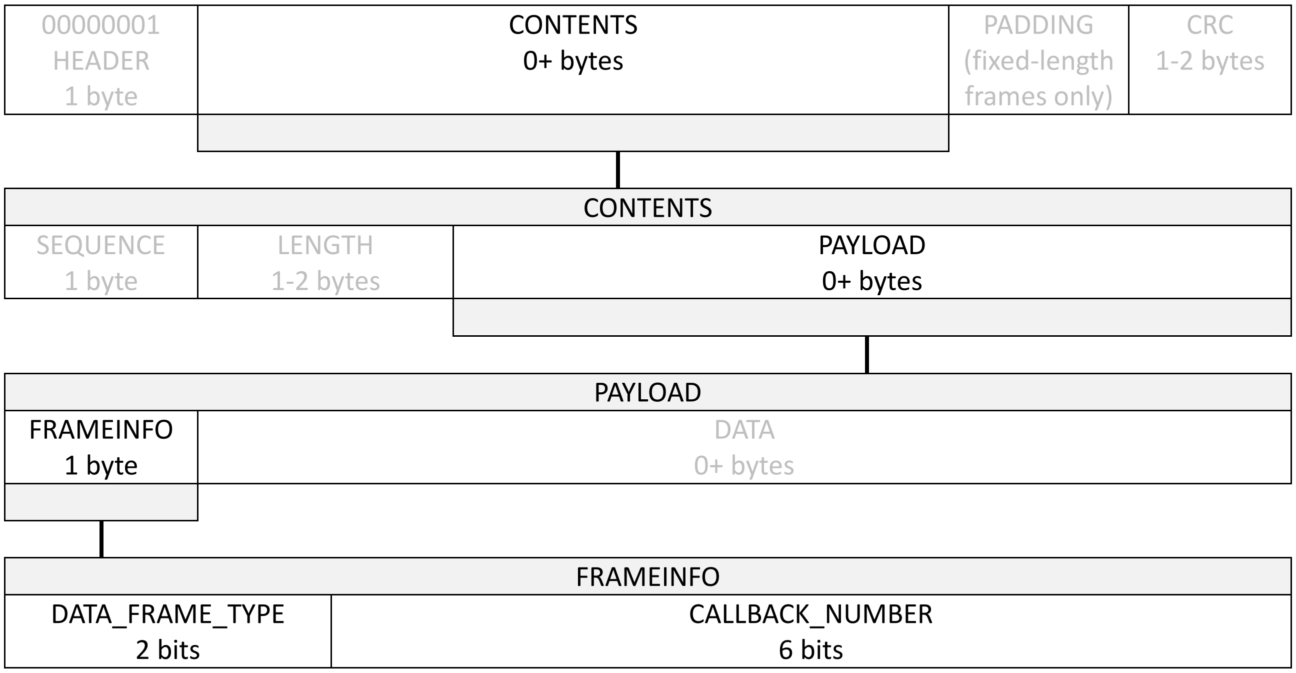
When the transport's CRC is set to CRC-8-CCITT, PAYLOAD_LENGTH is a 6-bit value (payload length of 0-59 bytes).

When the transport's CRC is set to CRC-16-CCITT, PAYLOAD_LENGTH is a 14-bit value (payload length of 0-16,377 bytes).

CONTENTS > PAYLOAD contains the PAYLOAD of the data frame.



CONTENTS > PAYLOAD > FRAMEINFO describes the type of DATA sent in the DATA frame and provides an optional CALLBACK_NUMBER which is used to correlate function calls with their responses/exceptions.



CONTENTS > PAYLOAD > FRAMEINFO > DATA_FRAME_TYPE has four options (mask: 0xC0, shift: >> 6):

- 0x0 = FUNCTION_CALL
  Direction: MASTER -> SLAVE
- 0x1 = ASYNC_EVENT
  Direction: SLAVE -> MASTER
- 0x2 = FUNCTION_RESPONSE
  Direction: SLAVE -> MASTER
- 0x3 = FUNCTION_EXCEPTION
  Direction: SLAVE -> MASTER

FUNCTION_CALL is a command sent to the slave along with function-specific parameters.

ASYNC_EVENT is an asynchronous message sent to the master from the slave. This message is either a profile-specific asynchronous event such as “UART_DATA_AVAILABLE” or an interim status update from a function call.

FUNCTION_RESPONSE contains the response to a previously-called function. The response’s DATA will contain the function’s return value(s); if the function doesn’t return values then DATA will be empty.

FUNCTION_EXCEPTION indicates that a previously-called function encountered an error and could not complete. The exception’s DATA will contain the 32-bit exception number optionally followed by exception-specific parameters.

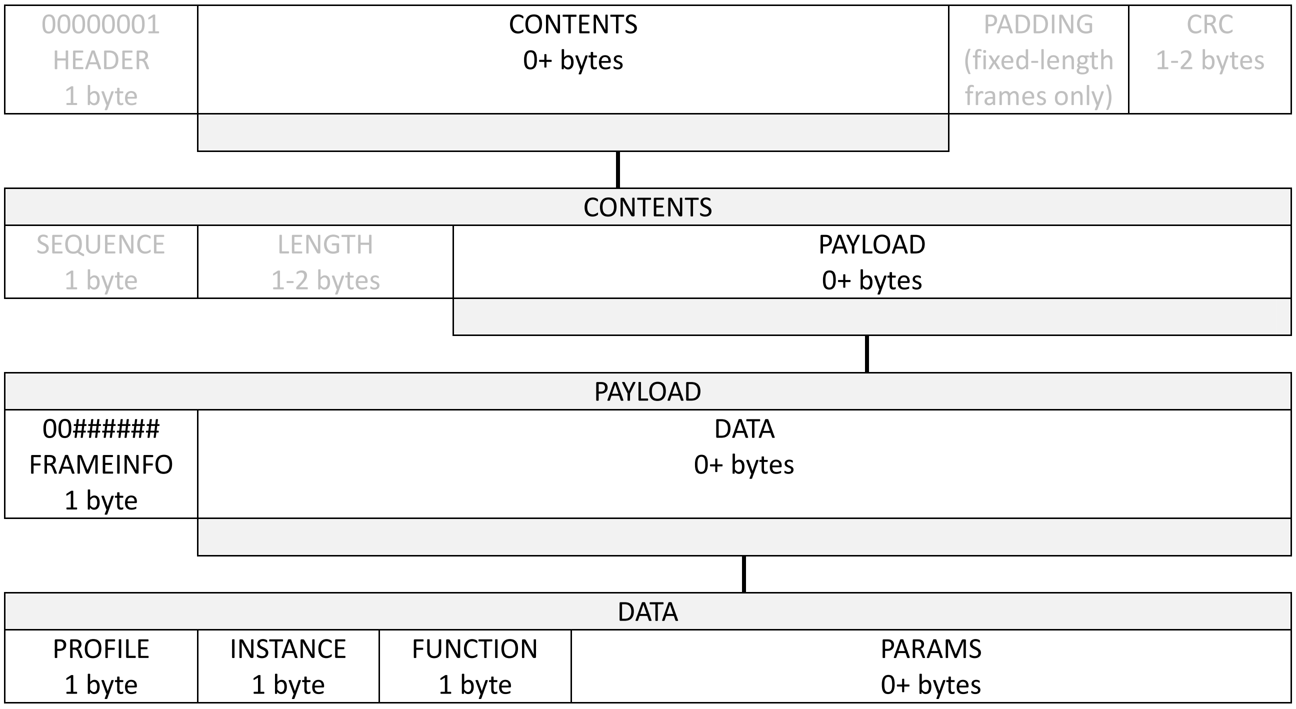
CONTENTS > PAYLOAD > FRAMEINFO > CALLBACK_NUMBER is a callback number used to correlate function calls with their responses/exceptions and to enable asynchronous updates from outstanding function calls. Up to 63 concurrent functions (1-63) may be outstanding at any time.

For function calls, if CALLBACK_NUMBER is zero, then the function is called in fire-and-forget fashion. In such a case no interim ASYNC_EVENT or final FUNCTION_RESPONSE or FUNCTION_EXCEPTION will be received and the caller will not be informed when the function completes. Standard function calls always use a non-zero CALLBACK_NUMBER but callbacks are optional for system function calls like NOP.

If an ASYNC_EVENT is raised which is not correlated with a currently-running function call, then the CALLBACK_NUMBER for that asynchronous event will be zero. An example of this is “UART_DATA_AVAILABLE” which is sent when a slave’s UART has received new data.

CONTENTS > PAYLOAD > DATA contains data which varies between DATA_FRAME_TYPE types.

Format for FUNCTION_CALL DATA



CONTENTS > PAYLOAD > FRAMEINFO contains the DATA_FRAME_TYPE of 0x0 followed by a 6-bit CALLBACK_NUMBER. The CALLBACK_NUMBER for function calls must be non-zero except in the case of system functions.

When CALLBACK_NUMBER is non-zero, function completion will be indicated through a FUNCTION_RESPONSE or FUNCTION_EXCEPTION callback.

CONTENTS > PAYLOAD > DATA > PROFILE is the profile ID for the FUNCTION. This is similar to a class or interface in object-oriented programming languages.

Some example profiles are:

- 0x10 = GPIO
- 0x11 = ADC
- 0x13 = PWM
- 0x14 = UART
- 0x15 = STREAM

Available profiles for a device may be enumerated through the DEVICE profile.

CONTENTS > PAYLOAD > DATA > INSTANCE designates the instance # of the profile which the slave should use to execute the requested FUNCTION.

The default INSTANCE for a profile is zero. Profiles such as the DEVICE profile support exactly one instance. Other profiles such as GPIO or SPI can support multiple instances of their profile. Up to 128 instances (0-127) of each profile are supported for a given device.

INSTANCE indicates the instance (0-127) of the specified profile to be addressed. As an example, a device may have 10 GPIOs. Many profiles have only one instance, in which case INSTANCE is 0. Profile instances are sequential, starting with INSTANCE 0.

The highest bit of INSTANCE is reserved, should be populated with zero, and should not be interpreted.

The number of available instances of each profile may be requested through the DEVICE profile.

CONTENTS > PAYLOAD > DATA > FUNCTION indicates the profile-specific function to execute on the specified instance of the specified profile. This is similar to calling a function of an object instance in object-oriented programming languages.

An example profile function call is:

- FRAMEINFO > CALLBACK_NUMBER: 0x33
  PROFILE: GPIO (0x10)
  INSTANCE: 2nd instance (0x01)
  FUNCTION: GPIO_READ (0x01)

CONTENTS > PAYLOAD > DATA > PARAMS indicates the profile function's parameters (if the function has parameters).

An example profile function call with parameters is:

- FRAMEINFO > CALLBACK_NUMBER: 0x34
  PROFILE: GPIO (0x10)
  INSTANCE: 2nd instance (0x01)
  FUNCTION: GPIO_WRITE (0x00)
  PARAMS: {0x00} <-- drive GPIO to low level (0V)

Format for FUNCTION_RESPONSE DATA

00000001 HEADER 1 byte	CONTENTS 0+ bytes	PADDING (fixed-length frames only)	CRC 1-2 bytes

CONTENTS		
SEQUENCE 1 byte	LENGTH 1-2 bytes	PAYLOAD 0+ bytes

PAYLOAD	
10###### FRAMEINFO 1 byte	DATA 0+ bytes

DATA
RETURN_VALUES 0+ bytes

CONTENTS > PAYLOAD > FRAMEINFO contains the DATA_FRAME_TYPE of 0x2 followed by the 6-bit CALLBACK_NUMBER corresponding to the function response. CALLBACK_NUMBER will never be zero since FUNCTION_RESPONSE data frames are only sent to report completion of function calls.

CONTENTS > PAYLOAD > DATA > RETURN_VALUES indicates the completed profile function's return values; if the function does not have return values then RETURN_VALUES will be empty (0 bytes).

An example profile function call and function response is:

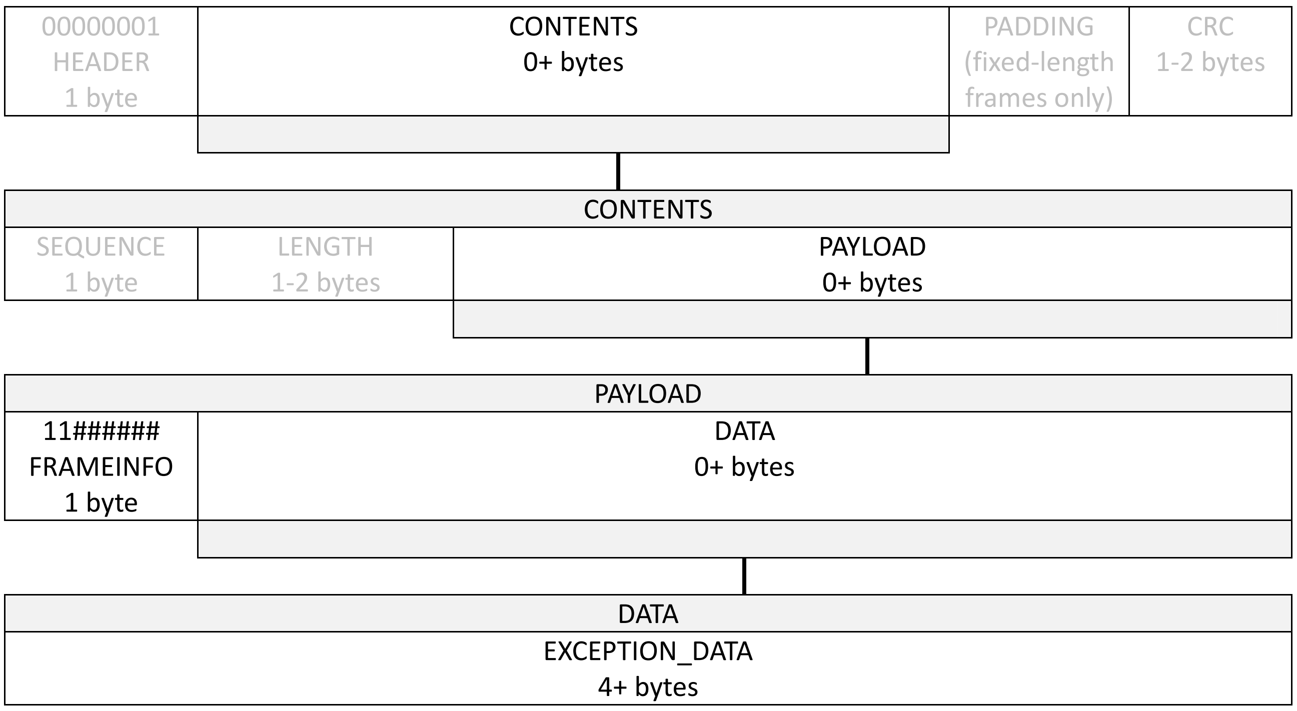
FUNCTION_CALL

- FRAMEINFO > CALLBACK_NUMBER: 0x35
  PROFILE: GPIO (0x10)
  INSTANCE: 2nd instance (0x01)
  FUNCTION: GPIO_READ (0x01)

FUNCTION_RESPONSE

- FRAMEINFO > CALLBACK_NUMBER: 0x35
  RETURN_VALUES: {0x00} <-- GPIO is currently at low level (0V)

Format for FUNCTION_EXCEPTION DATA



CONTENTS > PAYLOAD > FRAMEINFO contains the DATA_FRAME_TYPE of 0x3 followed by the 6-bit CALLBACK_NUMBER corresponding to the function exception. CALLBACK_NUMBER will never be zero since FUNCTION_EXCEPTION data frames are only sent to report exceptions thrown by functions.

CONTENTS > PAYLOAD > DATA > EXCEPTION_DATA indicates the aborted profile function's exception value; if the exception has parameters then these follow the exception number.

An example profile function call and function exception is:

FUNCTION_CALL

- FRAMEINFO > CALLBACK_NUMBER: 0x36
  PROFILE: UART (0x14)
  INSTANCE: 1st instance (0x00)
  FUNCTION: UART_READ, UP TO 4 BYTES (0x01, 0x04)

FUNCTION_EXCEPTION ("no bytes received within READ_TIMEOUT")

- FRAMEINFO > CALLBACK_NUMBER: 0x36
  EXCEPTION_DATA: {READ_TIMEOUT_EXCEPTION}

Format for ASYNC_EVENT DATA

ASYNC_EVENT data frames use the FUNCTION_CALL data format when asynchronous events are being transmitted (such as UART_DATA_AVAILABLE). This is indicated by a CALLBACK_NUMBER set to zero.

ASYNC_EVENT data frames use the FUNCTION_RESPONSE data format when reporting progress updates to an outstanding function call. This is indicated by a non-zero CALLBACK_NUMBER.

gobus frame transport

<u>GOBUS SPI TRANSPORT</u>

The traditional method for using GoBus uses GoPort IDC headers and GoCable IDC cables.

<u>GoPort IDC header pin-out</u>

SPI_CLK PIN 9	SPI_MOSI PIN 7	UART_TX* + /RESET PIN 5	GPIO + /INT + BOOT PIN 3	3V3 PIN 1
GND PIN 10	SPI_MISO PIN 8	/SPI_SS PIN 6	UART_RX* + SWIM PIN 4	5V PIN 2

* UART_TX/RX are from the module's perspective

The GoPort header provides:

- 3V3 and 5V power supplies
  - 20mA of 3V3 (or 12mA of 5V) at power-up; more can be requested post-enumeration
- GND pin (common GND)
- /INT + BOOT
  - GPIO is used with non-MCU modules (button, LED, etc.) as general input or output
  - /INT is asserted by MCU-based modules to signal that module has data to send and is de-asserted once all data has been transmitted
  - During power-up of MCU-based modules, the master can hold this pin high (as BOOT) to put the module into boot loader mode for re-flashing
- UART_RX+UART_TX and SWIM+/RESET
  - Used for reprogramming modules (or as a debug channel during module development)
- /SPI_SS
  - Asserted by master to signal start-of-frame and begin exchanging frames with module
- SPI_MOSI and SPI_MISO and SPI_CLK
  - Data channel for GoBus traffic

All data signals are 3V3 high, 0V low.   5V signals are not permitted on GoBus data lines.

<u>Initial transport settings</u>

- 18-byte fixed length frame
- Frame window size: single-frame
- CRC8

Exchanging frames

When the master asserts /SPI_SS, the master and slave begin exchanging frames. The slave should queue up data frames so that they can be sent immediately upon first SPI_CLK after /SPI_SS assert. The slave shall queue up an empty frame (all zeros) when there are no frames ready to send.

Sent and received frames are identical lengths, using the fixed frame format. PADDING is inserted between the PAYLOAD and CRC so that DMA can be used to send/receive SPI traffic and calculate/verify the CRC values.

GOBUS ASYNCHRONOUS TRANSPORT

GoBus can also be used via asynchronous transports such as UARTs (including XBee Serial and Bluetooth SPP).

GoBus is designed to work with a variety of transports of varying latency characteristics. There are no physical wiring diagrams provided for asynchronous transport as it is media independent.

When using asynchronous transport, no flow control mechanisms outside of GoBus are required. The slave can send messages to the master at any time and the standard ACK sequence counter will handle flow control requirements.

If using non-error-corrected RF for asynchronous transport, the transport should switch to CRC16 mode.

Start-of-Frame Byte

Since asynchronous transports can experience loss of frame synchronization, an additional byte is prepended to each GoBus frame. This byte is not included in CRC calculations. This start of frame byte (SOF) has the value 0xA5.

SOF (0xA5) 1 byte	HEADER 1 byte	SEQUENCE 1 byte	LENGTH 1- 2 bytes	PAYLOAD (0+ bytes)	PADDING (fixed-length frames only)	CRC 1- 2 bytes

Frame Timeouts

When bytes are received via asynchronous transport, the receiver will search for a SOF byte. Once that byte is found and the LENGTH value is received, the receiver will then wait for the remaining frame bytes to be received.

Asynchronous transport mechanisms must have a TIMEOUT setting:

- If no additional bytes are received after TIMEOUT milliseconds (i.e. the frame is not complete), then the frame will be considered corrupt. The SOF byte will be discarded and the receiver will send a NAK frame and search for another SOF byte.
- For asynchronous transports with external reconnection mechanisms, the TIMEOUT can be set to 0 and as such will be ignored. In the case that the master needs to resync with the slave, it can reset its connection and send a transport-only RESET_REQUEST instead.

gobus profiles

The following profiles are samples only; full profiles are TBD.

TRANSPORT PROFILE (0x00)

0. Get Supported Frame Format Capabilities
   requests maximum packet length, maximum frame size , # of outstanding frames, CRC16 support
1. Set Frame Format
   sets maximum packet length, maximum frame size , # of outstanding frames, CRC16 support
   – format changes as soon as call is ack'd, which will come via the previous frame format.

DEVICE (MANAGEMENT) PROFILE (0x01)

0. Get Version (major as byte, minor as byte)
1. Get Supported Profiles (as byte array)
2. Get Instance Count of Profile
3. Get Manufacturer Name String (optional)
4. Get Product Name String (optional)

STM32 FLASH PROFILE (0xD0) -- temporary sample, to be replaced with generic flash profile

OPEN PORT FOR FLASHING

0xD0 PROFILE	0x00 INSTANCE	0x00 COMMAND	PORT # (1 byte)

CLOSE PORT

0xD0 PROFILE	0x00 INSTANCE	0x01 COMMAND

GET CHIP TYPE

0xD0 PROFILE	0x00 INSTANCE	0x02 COMMAND

READ MEMORY REQUEST

0xD0 PROFILE	0x00 INSTANCE	0x03 COMMAND	SECTOR # (1 byte)	PAGE # (2 bytes)	# OF BYTES (2 bytes)

[READ MEMORY REQUEST] RESPONSE

FLASH DATA (# OF BYTES bytes)

WRITE MEMORY

0xD0 PROFILE	0x00 INSTANCE	0x04 COMMAND	SECTOR # (1 byte)	PAGE # (2 bytes)	# OF BYTES (2 bytes)	FLASH DATA (# OF BYTES bytes)

ERASE ALL

0xD0 PROFILE	0x00 INSTANCE	0x05 COMMAND

ERASE PAGES

0xD0 PROFILE	0x00 INSTANCE	0x06 COMMAND	SECTOR # (1 byte)	# OF PAGES (2 bytes)

EXECUTE AT ADDRESS

0xD0 PROFILE	0x00 INSTANCE	0x07 COMMAND	ADDRESS (4 bytes)

module drivers (for .net micro framework)

NETMF LIBRARIES (C# DRIVERS)

NETMF drivers (support libraries) for GoBus modules must inherit from the GoBus.GoModule class and must implement the standard constructor pattern.

Constructor 1:
new Class();

Constructor 2:
new Class(GoBus.GoPort port);

If the module driver needs more parameters, append them to the constructor params:

new Class(string param1); // auto-discover module, pass in param1 as string

new Class(GoBus.GoPort port, string param1); // use specific port, pass in param1 as string

SOURCE CODE

NETMF drivers must be provided in source-code format. Modules are not required to use open source firmware, but source code for drivers is critical to enable users to use drivers with future versions of NETMF.

STANDARD PROFILES

GoBus modules implement standard profiles to expose basic microcontroller peripheral features.

For example, GPIOs exposed to a user must be wrapped in the GPIO profile (which can then be used on the mainboard as InputPorts, OutputPorts, InterruptPorts, and/or TristatePorts using the virtual Cpu.Pins of the GPIO instances).

Users should be able to access GPIOs using the standard NETMF Port classes; users should be able to access ADCs using the standard NETMF AnalogInput class; users should be able to access an exposed SPI channel using the standard NETMF SPI classes; etc.

Standard profiles include but are not limited to GPIO, ADC, DAC, PWM, UART, SPI, and I2C.

GPIOs that cannot be directly manipulated by the user or NETMF driver (including GPIOs used for GoBus power management) should not be exposed as virtual i/o.

The module creator then wraps the virtual i/o in a higher-level driver class. As an example, a GPS module driver could wrap the NMEA data stream (received through a virtual SerialPort UART) in high-level events and properties. This also gives advanced users the option to query the GoBus framework for the underlying UART and instead read the NMEA data stream directly.

If the module has onboard logic which needs to read/write custom data which does not map to a standard microcontroller peripheral feature (i.e. there is no applicable GoBus profile), the STREAM profile can be used.

glossary

CRC is cyclic redundancy check. This error-detecting check value is appended to each frame to help detect data corruption during frame transmission. Two standard CRC algorithms, CRC-8-CCITT (CRC8) and CRC-16-CCITT (CRC16), are supported.

FRAME WINDOW SIZE indicates the number of data frames that can be transmitted before requiring acknowledgement by the receiver. A single-frame window is the default setting and indicates that each frame must be acknowledged before another frame is transmitted. A multi-frame window allows multiple frames to be transmitted before requiring acknowledgment; since ACKs can take a few frames to be returned to the transmitter this can speed up communication dramatically.

FUNCTION is a PROFILE command which executes some action (similar to a class method in an object-oriented programming language). For example a UART will have a READ function which retrieves its incoming data and a WRITE function which puts data in its outgoing buffer.

INSTANCE designates which instance of a PROFILE feature is being referred to. For example a device may have 10 GPIO pins. These GPIO pins are GPIO INSTANCE 0 through GPIO INSTANCE 9.

INT is the interrupt pin used by SPI transport. By asserting this pin (i.e. driving it to GND) the slave device indicates to its master that it has pending data frames to transmit.

MUTLI-FRAME WINDOW – see FRAME WINDOW SIZE

PACKET is the concatenation of one or more DATA FRAME transmissions. Some profile functions may have parameters which are too large to fit in a single frame. In these cases, multiple frames can be combined into a larger packet. If all parameters fit into a single frame, then that frame's data comprises the entire packet.

PROFILE is a set of commands (similar to a class or interface in an object-oriented programming language). Each command is called a FUNCTION.

SINGLE-FRAME WINDOW – see FRAME WINDOW SIZE

gobus trademark usage

Interoperability of mainboards and modules is central to the GoBus philosophy.

The GoBus interoperability logo is licensed royalty-free to module builders to indicate interoperability of modules with Netduino Go and the GoBus standard.

To use this logo on or to market your products as GoBus compatible, submit your GoBus compatible modules to Secret Labs for logo license approval. Secret Labs is developing a set of self-certification guidelines and a "click-through" license agreement to streamline this process in the future.

Mainboard manufacturers may also incorporate GoBus technology into their mainboards and obtain a royalty-free license to market their products as GoBus compatible. Please contact Secret Labs for details and logo license approval.

legal

Copyright © 2012 Secret Labs LLC. All rights reserved.

This work is licensed under a Creative Commons Attribution-NoDerivs 3.0 Unported License.

GoBus and the GoBus interoperability logo are registered trademarks of Secret Labs LLC.

GoBus is an open standard. No royalties are due to Secret Labs for use of the GoBus protocol in commercial products. To preserve interoperability and compliance with the GoBus protocol, a royalty-free license from Secret Labs LLC is required to use the GoBus trademark or GoBus interoperability trademark.