Guideline 5 Embedded Remote Procedure Call Protocol (*e*RPC)

# Purpose of the protocol

The purpose of the RPC communication protocol is to build a stable way of communicating independently of the communication protocol and the nature of the client. In this way multiple clients can communicate independently of their implementation and the communication environment (Ethernet, USB, etc.) . The protocol permits bidirectional data and command transfer between the server and the client(s). The protocol applies to embedded systems and it will be henceforth referred to a embedded Remote Procedure Call (*eRPC*) protocol. Not to be confused with the NXP eRPC protocol [1] which is centered on communication between multi-core designs.

# Protocol specification

eRPC is based on well-formatted messages that are bidirectionally passed between peers, independently of the physical link.

An eRPC message is composed of a message header and a message body:

M[k] = [ H[n]; B[m]]

Where M[k] is the message, H[n] is the header, B[m] is the body, n is the header length, m is the message body length and k = n+m is the message length.

The message M is composed of sequentially organized 32-bit words of varying length. The upper limit of the message length is the maximum value of the 32-bit unsigned integer type.

eRPC defines two types of messages: *function call messages* and *data messages*. The general structure of the function call messages (Call, Response, Fault,Event) is described in Figure 2, and the structure of the Data message is depicted in Figure 3.

One drawback of the protocol is that the message M does not contain all the information required to interpret the message contents. A properly documented message structure needs to be available such that client applications can properly be implemented against expected parameter types. However, this has the advantage of faster processing since there is no need for additional steps to decode type information embedded in the message. In addition, a lack of type specification in the message contents saves transmission bandwidth therefore further improving performance.

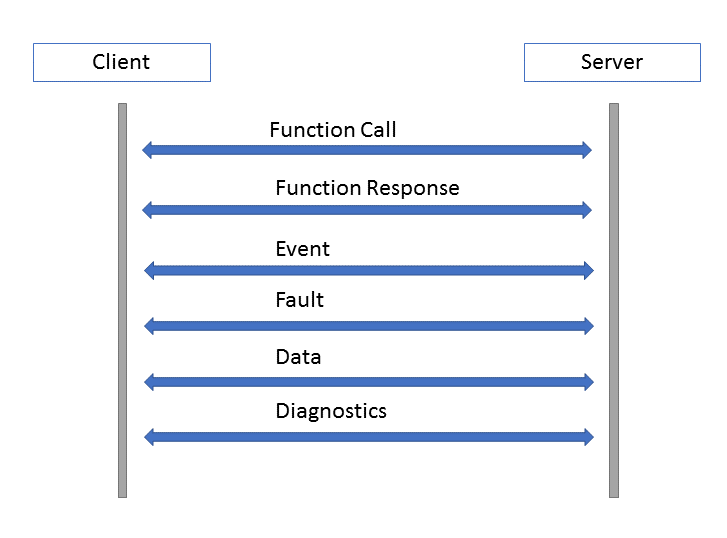


Figure . The eRPC protocol is bi-directional and independent of the physical layer.

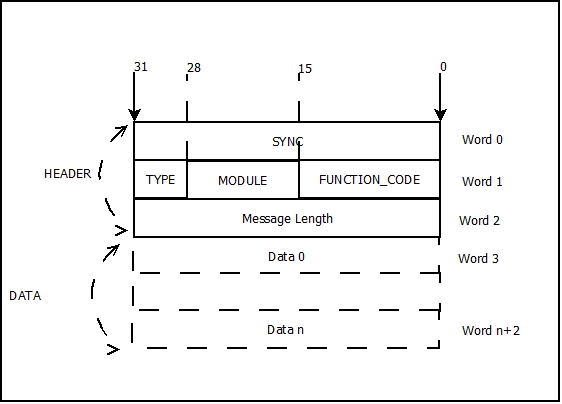


Figure . Function call message.

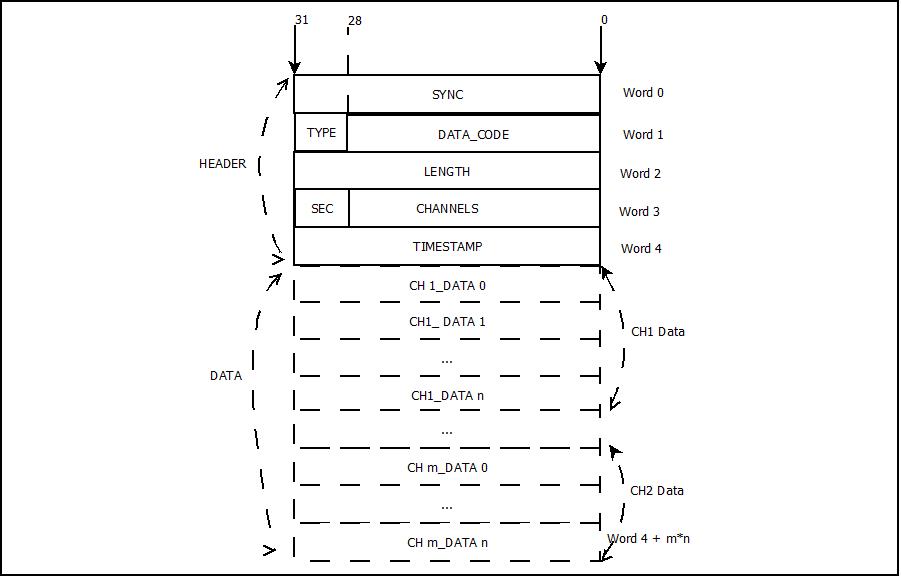


Figure . Data message.

## Message Structure

Based on the message type there are two types of headers:

### Base Header

The base header is composed of a SYNC word, a message type word and a message length word and is depicted in Figure 1.

All messages start with a SYNC code, with a predefined value of 0x10000001. This is used to identify the beginning of each message and avoid de-synchronizing the message parsing loops.

The base header begins with the SYNC code on Word 0, followed by information pertaining to the specific message type, module and function to call:

* TYPE- 4 bit code, specifying the type of the message. (See Table 2‑1)
* MODULE- 12 bit code. This code identifies the module or the logical group which contains the function. Each module implements a function call handling routine, such that there is transparency between modules concerning their specific functionality. Module codes are user/ programmer defined based on specific application implementation.
* FUNCTION\_CODE – 16 bit code. Each function is assigned a unique code inside its module.
* Message Length: The number of words that follow.
* DATA: Data content of the message. For function calls, the parameters of the function are transferred in the DATA words.

Table ‑. eRPC message type codes.

|  |  |
| --- | --- |
| Message Type | Code |
| Call | 0xC0000000 |
| Data | 0x30000000 |
| Response | 0xA0000000 |
| Fault | 0x50000000 |
| Event | 0x90000000 |
| Diagnosis Code | 0x70000000 |

### Data Header

The Data message has the following structure (Figure 2):

* SYNC- is a unique code use to identify RPC messages.
* TYPE- 4 bit code, specifying the type of the message. (See Table 2‑1. eRPC message type codes.)
* DATA\_CODE- 28 bit code specifying the data type found in the message. (ex. Different codes for EEG/EKG/EMG/MER data ).
* LENGTH- the length of DATA section.
* CHANNELS –a mask identifying the input channels. Each bit specifies the state of an input channel. If the bit is set to 1, then data for that channel is found in the DATA section.
* SEQ- Is the sequence or the window. For devices with more than 28 channels, the channels will be grouped in sequences or windows and for each group of channels a data message with the same timestamp will be sent. The channel number is obtained by multiplying the sequence value with the corresponding channel position in the CHANNELS bit map. (ex. for 128 channels, 5 windows will be created for all 128 channels and 5 messages with the same timestamp will be sent).
* TIMESTAMP- message timestamp used to recreate the data sequence on the receiving end.
* DATA – The DATA section contains samples for channels specified in the CHANNELS field. Within the DATA section, each channel occupies its own subsection, as depicted in Figure 2.

## Message Types

### Function Call

This message type sends the function code and the code of the module where the function is implemented and the list of parameters from left to right (Figure 4). The number of parameters is specified in Word 2 of the message header. No type code is specified for the parameters but casting or un-packing is done on the client/ server side according to the function parameters’ types.

The default type of the transmitted data is 32-bit unsigned int. For other numeric data types, such as floating point, a type conversion from float to unsigned integer is performed during packing and then, during un-packing the data is reconverted from unsigned int to floating point.

When structures are sent, the structures are serialized and packed in 32 bit words. Padding with zeros in the last word must be used, to keep the memory alignment of structures holding different data types. Only structures that have previously been – implicitly or explicitly – serialized in memory can be sent this way.

Integer type arrays will pack each element on a 32 bit word. Arrays of structures are supported, but require extra care because of element alignment issues.

The function call parameters: Param 0 - Param n will be send in the exact order as they are defined in the function definition, from left to right. If the parameter is an array or a structure the elements will be serialized and packed in 32 bit words without any type specification, as described above.

If a parameter is shorter than a 32-bit word, it will be padded with zeros to the left and occupy the full word.

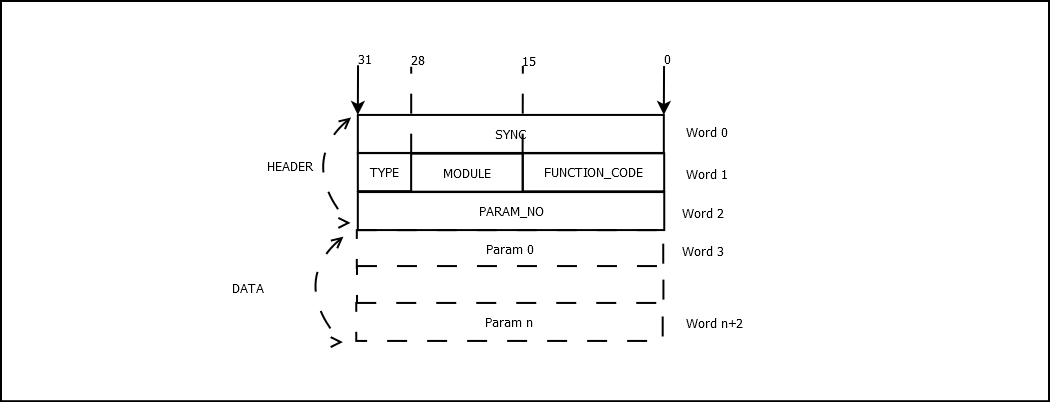


Figure Function Call message

For function call messages, the Message Length word (Word 2) specifies the number of function parameters found in the DATA section of the message.

* SYNC- is a unique code use to identify RPC messages.
* TYPE- 4 bit code, specifying the type of the message. (See Table 1)
* MODULE- 12 bit code. This code identifies the module or the logical group which contains the function. Each module implements a function call handling routine, such that there is transparency between modules concerning their specific functionality. Module codes are user/ programmer defined based on specific application implementation.
* FUNCTION\_CODE- every function is assigned a unique binary code
* PARAM\_NO –The number of function parameters
* PARAM 1-PARAM n the values of function parameter regardless of their type with length of 32 bits.

Module and function codes must be published for all entities involved in the communication involving the protocol specified here.

## Response Message

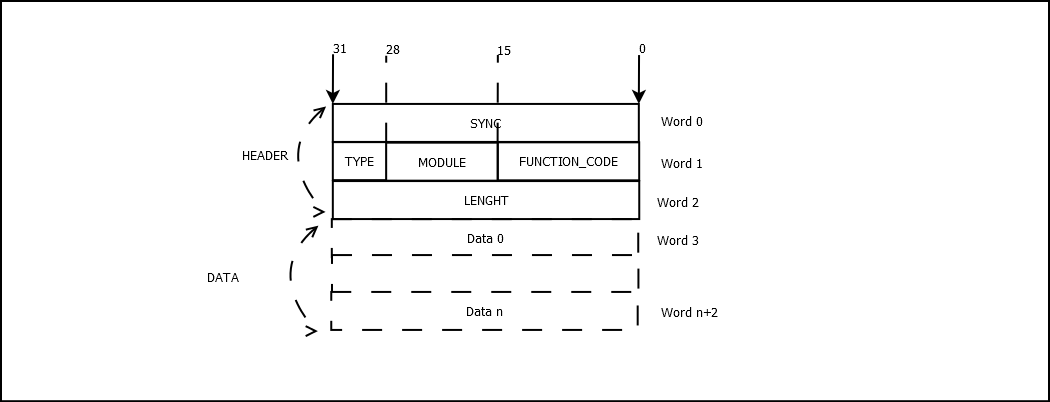


Figure Response Message

A Function Call message is usually followed by a response message sent by the callee.

The structure is the same as for the function call message.

The response message specifies the MODULE and FUNCTION\_CODE of the function that this message is in response to. These values are taken from the Function Call message.

The DATA section contains response information such state variables, serial numbers, etc.

## Fault Message

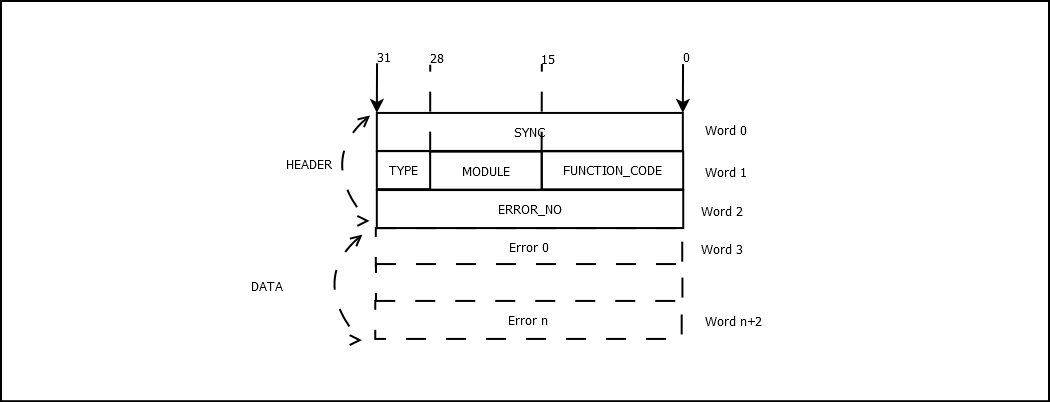


Figure Fault Message

The Fault Message will be sent as a response to a function call that failed to execute properly.

The Fault message specifies the MODULE and FUNCTION\_CODE of the function that caused the failure along with the error code or codes in the DATA section. The number of error codes is specified in Word 2.

## Event Message

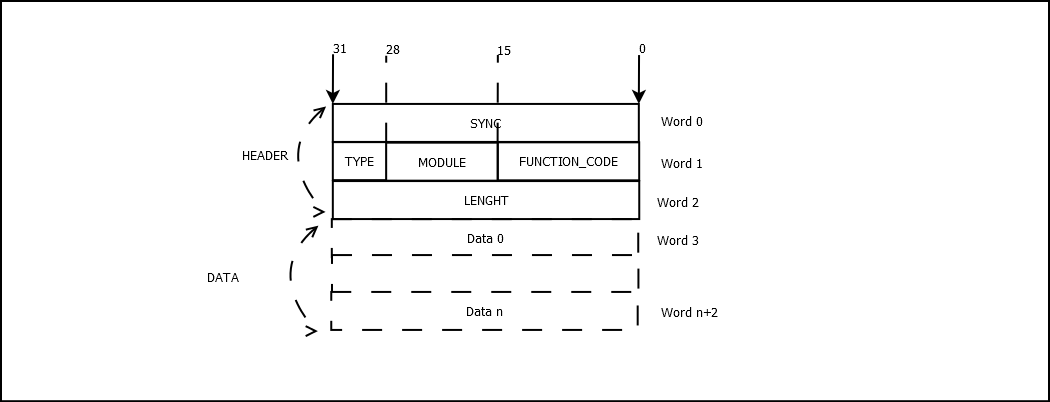


Figure 7 Event Message

An event message is sent as a result of an external or internal event pertaining to the client or server that was not otherwise triggered by a function call in the sense defined in this document.

The event messages share the same general structure common to all function call messages.

The event message contains a type specifier, the module where the event was initiated from and the event code. The reset of the message contains information on the length and data associated with that event.

The data type transmitted by the event message includes integers and single precision floating point values. These values are interpreted based on the event documentation.

## Data Message

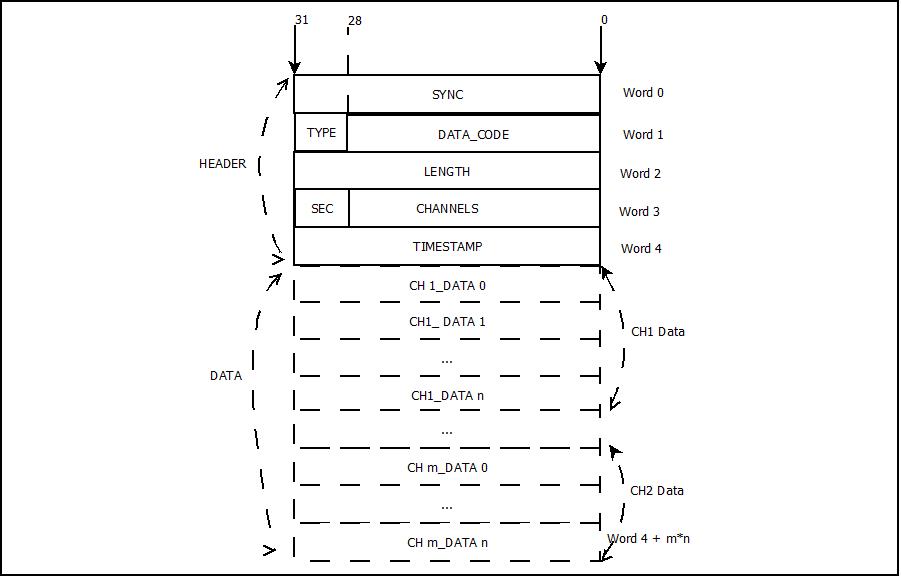


Figure Data Message

The Data message has the following structure:

* SYNC- is a unique code use to identify RPC messages.
* TYPE- 4 bit code, specifying the type of the message. (See Table 1)
* DATA\_CODE- 28 bit code specifying the data type found in the message. (ex. Different codes for EEG/EKG/EMG/MER data ).
* LENGTH- the length of DATA section.
* CHANNELS –a mask identifying the input channels. Each bit specifies the state of an input channel. If the bit is set to 1, then data for that channel is found in the DATA section.
* SEQ- Is the sequence or the window. For devices with more than 28 channels, the channels will be grouped in sequences or windows and for each group of channels a data message with the same timestamp will be sent. The channel number is obtained by multiplying the sequence value with the corresponding channel position in the CHANNELS bit map. (ex. for 128 channels, 5 windows will be created for all 128 channels and 5 messages with the same timestamp will be sent).
* TIMESTAMP- message timestamp used to recreate the data sequence on the receiving end.
* DATA – The DATA section contains samples for channels specified in the CHANNELS field. Within the DATA section, each channel occupies its own subsection, as depicted in the figure.