M.A.R.T.Y. Middleware Protocol - v2.0

November 10, 2023

This page documents the latest version of the M.A.R.T.Y. Middleware Protocol (MMP). Note that MMP version 1 can be seen as a 'subset' of MMP version 2. They can be compatible, however new components in MMP 2 (such as variable payload count) cannot be used to talk to MMP 1 clients.

The M.A.R.T.Y. Middleware Protocol is a framework for all of the M.A.R.T.Y. bundled applications to communicate with each other. Note that it is a **messaging** protocol, not a **file transfer** protocol. As such, it does not necessarily care about supporting completely arbitrary payloads, although it is designed in such a way that arbitrary payloads are possible with various encoding methods.

MMP version 2 has been modified with lessons from the MVP. Making the interactions between clients tighter and faster was of concern.

Changelog

- Packet size expanded from 1500 to 4096 bytes.
- SEND payload count changed from == 4 to >= 4, so that tokenising a message can be taken out of the application level.
- DELIVERY message type changed to APPLICATION and expanded to consolidate validation of application-layer operations, and avoid extra messaging.
- INVALID -> NOT_FOUND relocated to APPLICATION -> DELIVERY error.

Contents

Protocol Specification

1. Overview

An MMP packet consists of the following:

1. Message type *t*
2. Metadata Argument 1 *a1*
3. Metadata Argument 2 *a2*
4. Payload/s *p*
5. Payload Termination *e*
6. Message Termination *k*

The MTU of MMP packets is **4096 bytes**.

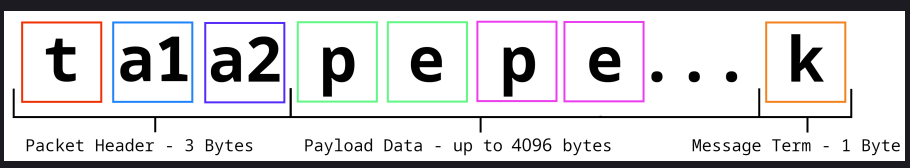
The **message header** is consisted of a **type *t*** and **argument bytes *a1*** and ***a2***, whose meaning depend on the type. It is a **set-size** of **3-bytes**. There can be ***n* number** of **payloads *p***, where the exact number *n* is **determined by** the **message type *t*** and **direction of transmission**.

NOTE: Payloads are terminated with the byte **0xFE** and packets are terminated with the byte **0xFF**. "Empty" data in MMP is designated with **0x00**, the ASCII NULL character. For example, an empty payload would be sent as **0x00FE**.

NOTE: The minimum payload count is determined by the type and direction, and is always the same. The meaning of payloads is determined by the arguments. Some message types can send a variable number of payloads.

With those special values excluded, the protocol allows for **252** message types, payloads to cover the **entire ASCII character range**, and the collection of 2 bytes for arguments (excluding termination values) allows for up to **63,504** possible argument values **per type**.

A standard packet breakdown:



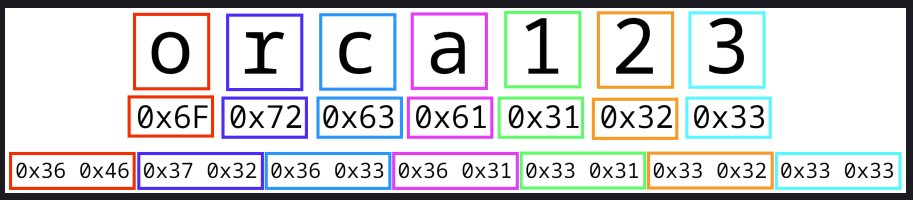
The **INIT** message type optionally allows for an **authentication token**. You can use it or not, seeing as it's just the base protocol auth method you can always implement a higher-level authentication method between your own clients when they've all connected using **SEND** messages and payloads.

---

2. Arbitrary Payloads

An example encoding method which would allow for **arbitrary payloads** is to encode the **raw hexadecimal data** as their **ASCII representation**, which fall into the supported character range. An example arbitrary payload encoding of the hex

representing the text ‘orca123’ can be seen below:



NOTE: It’s probably best to implement the encoding/decoding of this on the client-side rather than the server-side to avoid server overhead.

NOTE: If you’re only transmitting ASCII text you wouldn’t need to bother with an encoding style like this; the full ASCII range is supported in the base protocol.

---

3. Message Types

The current list of defined message types are detailed below.

Type	Byte	Description
INIT	0x01	Standard request for connection init
INVALID	0x02	Notify partner that the message type they sent was invalid
CONTERM	0x03	Notify partner that sender has terminated connection
SONAR	0x04	Ping partner to see if they’re still alive
SEND	0x05	Send a message to client(s)
APPLICATION	0x06	Inform partner of the outcome of application-layer events
QUERY	0x07	Query information about client(s)

NOTE: MMP has a **SONAR** message type, which is used as a periodic ping to see if a connection partner is still alive in both directions. The exact interval is up to the implementation; this document does not specify a standard interval.

NOTE: There is an **APPLICATION** message type that informs the connection partner of the outcome of a valid request. This will only be sent if the request was valid at a protocol-level and did not trigger an **INVALID** response.

A client uses the **APPLICATION** type in response to a **SEND** message. The server forwards a success or error (signified in the argument) to the sending client to notify it of the application-layer status, or if the target client does not respond at all.

---

4. Arguments

Argument behaviour is defined per message type. If a message type has no defined arguments, all messages transmit with that type **must have their arguments set to 0x00**.

4.1 INIT

Type	Arg	Byte	Description	Direction
INIT	INIT	0x0001	Client requests to initialise	Client -> Server
INIT	ACCEPT	0x0001	Init successful/accepted by server	Server -> Client

---

4.2 CONTERM

Type	Arg	Byte	Description	Direction
CONTERM	CLEAN	0x0001	Request a normal, clean connection termination	
CONTERM	SPAM	0x0002	Connection terminated due to too many invalid messages	
CONTERM	AUTH	0x0003	Connection terminated for authentication reasons	Server -> Client

---

4.3 INVALID

Type	Arg	Byte	Description	Direction
INVALID	TYPE	0x0001	Invalid *t*	
INVALID	LENGTH	0x0002	Message length before *k* exceeded max	
INVALID	ARGS	0x0003	Invalid *a* for *t*	
INVALID	PAYLOAD	0x0004	Invalid *p* for (*t* and *a*), e.g. data out of range	
INVALID	PAYLOAD_COUNT	0x0005	Invalid *n* for *t*	
INVALID	AUTH	0x0006	Authentication attempt was invalid	Server -> Client

Type	Arg	Byte	Description	Direction
INVALID	KEY_EXISTS	0x0007	Combination of requested name, priority and host already exist	Server -> Client
INVALID	RESERVED	0x0008	Reserved for future use	
INVALID	MESSAGE_ID	0x0009	Message ID was invalid	Server -> Client

---

4.4 SEND

Type	Arg	Byte	Description	Direction
SEND	DIRECT	0x0001	Send message to clients meeting specific criteria	
SEND	BROADCAST	0x0002	Send message to all clients	

4.5 APPLICATION

Type	Arg	Byte	Description	Direction
APPLICATION	SUCCESS	0x0001	Application operation completed successfully - everything else is an error	
APPLICATION	DELIVERY	0x0002	Delivery failed to complete	
APPLICATION	BUSY	0x0003	Client is busy with other requests	
APPLICATION	RANGE	0x0004	One or more bytes in the payload was out of range for app. layer	
APPLICATION	ENDPOINT	0x0005	The endpoint requested was invalid	
APPLICATION	VALUE	0x0006	One or more entire payloads were invalid	
APPLICATION	TOO_FEW	0x0007	Too few payloads provided for the endpoint	
APPLICATION	TOO_MANY	0x0008	Too many payloads provided for the endpoint	
APPLICATION	AUTH	0x0009	An application-layer authorisation error occurred	
APPLICATION	EXECUTE	0x000A	Request was technically valid, but execution error occured	

4.6 QUERY

Type	Arg	Byte	Description	Direction
QUERY	CONN_NUMBER	0x0001	Client requests total number of connected clients	Client -> Server
QUERY	CONN_NUMBER	0x0001	Server provides the total number of connected clients	Server -> Client
QUERY	CLIENT_INFO	0x0002	Client requests information about a client	Client -> Server
QUERY	CLIENT_INFO	0x0002	Server provides information about a client	Server -> Client
QUERY	NOT_FOUND	0x0003	Client's CLIENT_INFO request yielded no results	Server -> Client

---

5. Payload Configurations

Payload configurations are determined by the direction of communication and the argument.

NOTE: If there is no section here for a message type (e.g. INVALID), the packet of that type must be terminated with **0xFF** immediately after the arguments; **no payloads**.

NOTE: If the ***a*** column is left blank, the configuration applies to all arguments that aren't individually specified.

NOTE: In the *p* configuration column, if a payload field is left blank, it should be set to **0x00** before being terminated.

The meanings of the terms inside brackets:

- **req**: this is a required field
- **opt**: this is an optional field (if unspecified, set to **0x00** and terminate)
- **opt_r**: this is an optional field, however at least one of the payloads with this term must be present

5.1 INIT

Type	*n*	*a*	*p* Configuration	Direction
INIT	4		auth_token(**opt**),my_priority(**req**),my_host(**req**),my_name(**req**)	Client -> Server
INIT	4		,,,	Server -> Client

5.3 INVALID

Type	*n*	*a*	*p* Configuration	Direction
INVALID	1		,	

Type	*n*	*a*	*p* Configuration	Direction
INVALID	1	MESSAGE_ID	message_id(**req**)	

5.2 SEND

Type	*n*	*a*	*p* Configuration	Direction
SEND	>= 5	DIRECT	message_id(**req**),target_priority(**req**),target_host(**req**),target_name(**req**),message(**req**)…,message_n(**opt**)	Client -> Server
SEND	>= 5	DIRECT	message_id(**req**),sender_priority(**req**),sender_host(**req**),sender_name(**req**),message(**req**)…,message_n(**opt**)	Server -> Client
SEND	>= 5	BROADCAST	message_id(**req**),target_priority(**opt**),target_host(**opt**),target_name(**opt**),message(**req**)…,message_n(**opt**)	Client -> Server

---

5.3 APPLICATION

Type	*n*	*a*	*p* Configuration	Direction
APPLICATION	1		message_id(**req**)	

5.4 QUERY

Type	*n*	*a*	*p* Configuration	Direction
QUERY	3	CONN_NUMBER	,,	Client -> Server
QUERY	3	CONN_NUMBER	total_clients(**req**),,	Server -> Client
QUERY	3	CLIENT_INFO	target_priority(**opt**),target_host(**opt_r**),target_name(**opt_r**)	Client -> Server
QUERY	3	CLIENT_INFO	client_priority(**req**),client_host(**req**),client_name(**req**)	Server -> Client
QUERY	3	NOT_FOUND	target_priority(**opt**),target_host(**opt_r**),target_name(**opt_r**)	Server -> Client

---

It should be noted that this protocol document does not enforce any constraints about what a server considers “invalid”. It is up to the individual server to form and enforce these constraints. Behaviour when handling optional flags is also up to the implementation. Hoff Industries provides a server implementation known as *admiral*, but is not currently public for quality control reasons.

Message types and arguments are subject to change **at any time** with **no notice**. This page should be considered the **only** formal definition of the M.A.R.T.Y. Middleware Protocol.

Have fun with MMP!
