Mass Software Upgrade (MSU)

Implementation Guide

10/11/2014

Schneider Electric

Pawan Modi

1. **Abbreviation**

|  |  |
| --- | --- |
|  | Description |
| MSU | Mass Software Upgrade |
| SCM | Sequence Complain Mode |
| CCM | Chunk Complain Mode |
| CRC | Cyclic Redundancy Check |
| MAX\_SEQ\_LIM | Maximum Sequence Limit |

1. **Scope of the Document :**

The scope of this document is to give a direction to user who wants to implement MSU on their system. This document gives a complete flow of the MSU Protocol Specification V1.0 and also serve as a user guide taking into consideration many use cases.

1. **Introduction**

This document considers many use cases for individual message exchange between MSU Server and Client, and provides user more option to use MSU more efficiently and effectively on their platform/system.

1. **Network Discovery**

MSU is capable of discovering the devices connected on the network either within a specified device range or individually. MSU Sever sends the Discovery request [for message Format refer MSU Protocol Spec V1.0] on the network to achieve the same and all the MSU clients must respond to this request with the Discovery response. This Discovery response consist of two types of data

* Device Identification

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Size** | **Unit** | **Description** |
| Device ID | 4 | Byte | In caseIPV4 it can be Ip address of device or can be other 4 byte Identification number assigned by user |
| HW\_ID | 32 | Byte | String of HW ID |
| Product\_ID | 32 | Byte | String of Product description |
| Product\_Name | 32 | Byte | String of Product name |
| Model\_Name | 32 | Byte | String of Model name |
| Vendor\_Name | 32 | Byte | String of Vendor name/ID |
| FW\_SW\_Ver | 32 | Byte | FW/SW Version |
| Major\_Minor\_Revison | 32 | Byte | Major /Minor Version |

Note: Out of all above fields only Product\_Name and Model Name are optional.

* MSU protocol parameter

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Size in Bytes** | **Units** | **Description** |
| Sequence\_Delay | 2 | milliseconds | Delay between consecutive data packets |
| SCM\_Delay | 2 | milliseconds | Delay between consecutive SCMs |
| CCM\_Delay | 2 | milliseconds | Delay between consecutive CCMs |
| CCM\_Retry | 1 | number (0, 1,…255) | Number of CCM retries |
| SCM\_Retry | 1 | number (0, 1,…255) | Number of SCM retries |
| Timeout\_Val | 1 | seconds | Timeout value between any two consecutive response from the MSU server |
| Sequence Limit | 1 | Integer | Number of Sequence in a chunk Range Min 1 to Max 32 |

Note: All of the above fields are mandatory.

Protocol parameters are the mandatory parameters and have to be provided by the MSU Client. Each parameter can be derived by keeping in mind of packet processing time, memory available for temporary storage and file system operation time.

MSU server MUST be able to discover MSU Client connected on network either through “**One To Many**” or “**One To One”** datagram message.

* **One To Many**

MSU Server sends Multicast datagram message (Who-Is) with the start and end device ID ( detail refer to MSU Spec ) on the network and only those MSU Client can respond whose Device ID fall under this range others MUST ignore the message.

MSU Client responds with I-Am message using Unicast datagram message to the server.

MSU Server calculates the protocol parameter which will be used for future process unit next discovery command issued by it. To calculate the same, MSU Server considers the data contained within the I-Am messages received from Devices. Each device sends it protocol parameter within its I-Am message.

MSU server compares and picks protocol parameter from the devices by choosing slowest and lowest performance values.

 **Figure-1** MSU Network Discovery

To avoid collision on the network while sending I-Am message from device. Devices can put millisecond delay before sending message on the wire, and delay value would be the last byte of the device Id or IP address. In this case MSU Server can discover complete subnet by 255mSec.

There could be chances where MSU server/Client would not be able to receive either of the messages due to packet lost on the network during transmission. This is an unreliable way to discover the network and there are two ways to overcome this issue

1. MSU server can rediscover the entire network
2. MSU server can only discover the missing device

* **One To One**

MSU Server sends Unicast datagram message (Who-Is) to individual devices those falls in the range between start and end device ID (for details refer to MSU Spec).

MSU Client responds with I-Am message using Unicast datagram message to the server.

MSU Server calculates the protocol parameter as it does in **“One To Many”** case.

MSU Server can discover devices by the following 2 ways

* + - Send request and wait for response from individual device



* + - Send request to all device one by one and then wait for response from all devices



**Note** :To avoid network discovery on every upgrade process good to have one time and store locally on the server whenever need to start upgrade process import from the local storage and carry on the new process

1. **Connection Management and Authentication**

Except Scan/Discovery command all MSU commands are connection oriented. Server has to make prior connection with Client to access their MSU Feature. MSU maintains a single connection information for single process and every single connection information identified with unique transaction ID.

The Transaction ID must be a random number generated by MSU server and sent to the Client in the connect message with the other authentication information, once Client validate all authentication parameters successfully, Server and client can proceed with future messages exchange. Server and Client has to use same transaction ID for future communication any message from Server with wrong transaction ID Client assume authentication violation and will not entertain.

Connection message also includes user ID and password in their authentication parameters. Server must be authenticated by the Client with against its user ID and password, provided in connect message. Server can get access of Client only for the valid user ID and password else Client must not entertain any future communication from Server.



**Connect Message**

Client can define the multi level of user ID and Password authentication but has to inform to server. Client achieve same by providing Authentication bit field on I-Am message [ Ref MSU Protocol Specification V1.0].The configuration data is read from the bin file and the data in the bin file can be modified only if the user has admin rights. MSU protocol does not have way to edit and modify these configurations so user has to use same tool as it uses for configuration during installation(i.e. DTM, conf tools)

* Individual Level: every device will have unique “user name” and “Password”
* Group/subnet Level: single “user name” and “Password” for entire Group/subnet
* Disable user authentication: No User authentication required. In this case “User name and Password” for a client should be don’t care. Client expects NULL value for these two fields in the connection message.

Connection message is a two way message. Server sends a request to the Client and expects response. Once Client receive connection message from Server it validates/checks the integrity of the message and response with error code either ZERO (Successful) or Non-ZERO( Authentication failed)

Client also validates the Server IP with their IP White list. IP white list must be provided during network setup/commissioning/ installation through the “msu.bin” file.

Client can also validate file type/Name, vendor, H/W ID to make sure the file is transferred to right device and prevent from device corruption. These are option field in the connection message and need to decide by device whether they want to validate these parameters or not.

Once device successfully authenticates the server, it stores the connection information locally. After that Client only responds to only those messages that come from the authenticated Server. All the other packets are dropped.

This connection has Timeout period. If there is no activity on the line for this specified time, the following might happen

|  |  |  |
| --- | --- | --- |
| **Event** | **Description** | **Response** |
| Implicit Disconnect | If there is no activity at the server end, the server initiates an implicit disconnect, wherein the server generates a disconnect message and sends to all the Clients | Client responds with the Disconnect response message. For description refer the spec. file. |
| Explicit Disconnect | This can be used to consciously disconnect from the Client. Initiated and based on the discretion of the User. It can be called anytime before the Upgrade process starts based on the requirement. |
| Client Timeout | String of Product name |  |

The server sends an implicit timeout to the device once timeout occurs. Apart from this, anytime before the Upgrade process starts, the device can perform an explicit disconnect from the device. The disconnect message is followed by a disconnect response from the device. If there is any ambiguity in the connection, such as an invalid server IP, incorrect transaction ID or even the authentication level, the device must not reset the connection, and generate a response to the server with the correct error response.



Server must send disconnect message to each of connected device to it once it finishes the process to make sure the process is completed secure and safe.



Disconnect message

All the configuration information MSU client consumes comes from a configuration file named “msu.bin”. it is a Binary format file with defined structures value common for MSU protocol.

1. **Upgrade Process**

Once the connection stage is over, the Server sends out the notification message. This message contains information pertaining to the file to be transferred, such as the file number, the number of chunks to be transferred etc.

In such a case, validation is performed against the following information.

|  |  |
| --- | --- |
| **Field** | **Action Items** |
| Transaction ID | It must be the same as that for the given upgrade cycle |
| File Name | Must be defined |
| File Size | Must not be zero |
| Chunk Count | It must not be zero |
| Number of Sequences | Must fall within the valid range i.e.  0<Seq. No<=MAX\_SEQ\_LIM\* |
| Sequence size | Sequence Size must not be zero |
| Multicast Address | This must not be zero |
| Multicast port | This must not be zero |

MAX\_SEQ\_LIM is the maximum number of sequences permitted. It is generally limited to 32. The sequence size and number of sequences permitted is generally decided based on memory limitations at the Client end.

If the notify message is not parsed successfully due to any ambiguity in the data packet the device must disengage from the upgrade process.

Once the notification parsing is successful, the device must configure itself to listen on the new socket.

After sending out the notify message the server waits for a predefined time for the clients to configure themselves with the new socket. Soon after this the sever starts sending the packets.

The server divides the packet into chunks and the chunks into to the permissible number of sequences based on the information contained in the discovery response. The sequence limit is generally set to 32. This is generally done at the time of package selection. The data pertaining to the number of chunks and sequences, the file name etc, is reflected on the GUI.

For data transfer, the server reads the data by the sequence, by the chunk.

All the sequences in the chunk are read and transferred.

Once all the packets in a chunk have been transferred the Server waits for a predefined wait time, called the SCM wait time, wherein it waits for client generated SCM messages.

The client generated SCM message contains information about the packets missed. SCM Message only sent by the device who has lost at least 1 packet/Sequence from current chunk.

The Server collates this data that it receives from all the devices participating in SCM and re-transmits the missed sequences one by one.

The same is done for all the chunks.

Once all the chunks have been transferred the server sends the Transfer complete message.

Once the client receives the transfer complete message it checks for missed chunks and if there are any missed chunks, it generates a CCM message and sends it to the server.

The server again collates this data and sends out the packets.

This process is repeated until all the devices have received the packets and the data has been assembled successfully by the client.

Upon receiving all the packets the client performs a CRC check on the assembled file to check the integrity of the data. If the CRC check passes, the Client sets its status to Upgrade passed.



**Complete Upgrade Process**

1. **Upgrade Status**

Upon receiving all the packets the client performs a CRC check on the assembled file to check the integrity of the data. If the CRC check passes, the Client sets its status to Upgrade passed.

Once the CCM is complete, the Server queries all the clients for their upgrade status through a status check message.

The Client’s response to this message contains the information pertaining to the status of the upgrade.

As aforementioned, every client post CRC verification sets the status of the upgrade process either as

1. Upgrade Passed
2. Upgrade Failed
3. Upgrade In Progress.

Upgrade passed implies that the CRC verification has passed

Upgrade failed is the Status by default. It implies that the data integrity could not be established and hence the upgrade process failed.

The last is the status to reflect that the upgrade process is still in progress.

Once the CRC verification is complete and the data transfer is complete the Client must prepare itself for the next upgrade process.

To achieve this, the device performs a loopback wherein it resets all the upgrade process parameters and listens on the default multicast address.



**Upgrade Status**