Atmosic Over the Air Update Service User Guide for ATM33 Wireless SoC Family

SUMMARY: This document provides the functional specification and user guide for the Atmosic Over-The-Air (OTA) Bluetooth Low Energy (BLE) server profile. The OTA profile can be integrated into an application to provide reliable application software updates. The updates can be delivered via a Bluetooth LE client operating on a remote central device.

Atmosic™

Table of Contents

1. Overview 6

2. Features 6

New to Version 1.3 6

3. Design 8

3.1 Architecture 8

3.1.1 Client Profile 8

3.1.2 Server Profile 9

3.1.3 Upgrade Processor 9

3.1.4 Storage Subsystem 10

3.1.5 RRAM 10

3.1.6 MCUBoot 11

3.1.7 Upgrade State 11

3.2 Upgrade Process Flow 11

3.2.1 Use of a CRC/Hash 13

3.2.2 Upgrade Flow Step Details 13

3.2.3 First Boot Watchdog 15

3.3 Messages 15

3.3.1 Types 15

Command 15

Status 15

Completion 16

Write Data 16

3.3.2 Ports 16

Mailbox 16

Bulk 16

3.3.3 Definitions 17

Message Format 17

Command Status 18

Command Completion 18

Cancel 19
Bluetooth LE Considerations 19
Message size limits 19
Pending Message Indication 20
Locking and Unlocking Sessions 20
Lock Session 21
Unlock Session 21
Query Information 21
Query Version Information 23
Write Data Start 24
Write Data Continuation 25
Copy Data 26
Last Write Status 27
Read Data 27
Get Programmed State 28
Erase Sectors 29
Set Upgrade Status 29
Reboot 30
3.4 Message Flows 31
3.4.1 Control 31
3.4.2 Write Data 32
3.4.3 Write Data Stream Mode 32
4. Developers Guide 35
4.1 Installing the profile 35
4.1.1 Board Identifier 35
4.1.2 Write Buffer Size 35
4.1.3 First boot timeout 36
4.1.4 Version String 36
4.2 Changing the Service UUID 36
4.3 Configuring the Slot Sizes 36
4.4 Cleaning Upgrades 37

4.5 Testing with a Non-Upgradeable Partition Scheme 37

Revision History 38

References 38

Glossary 38

List of Figures

Figure 1 - OTA System Block Diagram
Figure 2 - RRAM Subsystem Block Diagram
Figure 3 - Image Upgrade Flow Diagram
Figure 4 - Command/Status/Completion Flow Diagram
Figure 5 - Write Data Flow for Mailbox Port
Figure 6 - Write Buffer Fill with a Dropped Message
Figure 7 - Write Data Flow for Bulk Port

List of Tables

Table 1 - Attribute Definition
Table 2 – Message Format
Table 3 – Status Codes

1. Overview

Over the Air (OTA) updates allow end users to update their systems in the field without the need for wired connections. A mobile application can deliver firmware updates wirelessly as part of the device onboarding process or a regularly scheduled update process. This document describes the Atmosic proprietary OTA update BLE profile (server/peripheral role) and messaging interface to allow customers to manage upgrades through mobile or desktop client applications. This document pertains to the unique use requirements when using internal RRAM memories available in ATM33 designs. Previous generations such as the ATM2/ATM3 series can continue to use the previous OTA protocols.

2. Features

- OTA server profile with customizable 128-bit service UUID.
- Server profile can coexist with other profiles.;Profile can be installed dynamically or statically at boot time.
- Low memory footprint with application configurable write buffer size.
- Supports Atmosic MCUBoot loader and upgradeable image partitioning scheme.
- Support for optional first boot watchdog mechanism
- Flexible OTA command set to develop both simple and complex (background update) client applications.
- Allows resumed updates from connection or power cycling interruptions.
- Standard Bluetooth LE link security and authentication
- Methods to inspect, copy (transfer) and write customer data
- Optional performance mode for write data transfers using the DTS bulk port data flow (experimental)

New to Version 1.3

The upgrade protocol version 1.3 is largely backwards compatible with 1.2. The following section lists some key differences.

- Extension to Query Information
  - Can indicate a small memory model to convey the partition size.
  - Can indicate that the storage device does not require an erase.
  - Can indicate that the storage device uses MCUBoot.
  - New flash type (RRAM).
- Boot Flow Changes

- Erase is now optional when the device reports this capability.
- For this BETA release, NVDS copy is not required or supported.
- Marking a bank as PROGRAMMED is not required when. MCUBoot is used.
- Marking a bank as BAD is not supported when MCUBoot is used.

3. Design

3.1 Architecture

The OTA system is described in Figure 1:

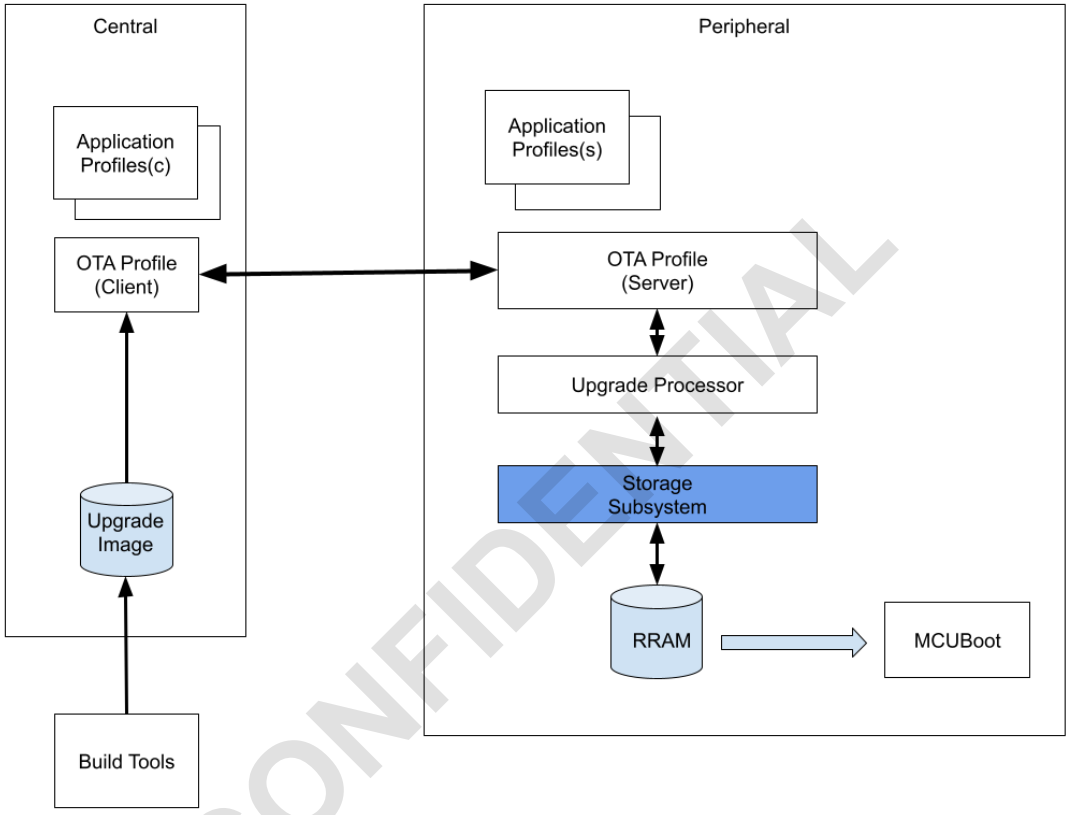


Figure 1 - OTA System Block Diagram

The central is the mobile or desktop host running the end-user application. Atmosic provides client libraries for common operating systems to rapidly integrate OTA capabilities into your application. The following section provides details on each of the major components.

3.1.1 Client Profile

The client profile is implemented in the central's BLE stack (mobile or desktop stack). The profile is GATT- based and will use GATT write and read procedures to communicate with server-side characteristics. The client interface to the OTA system consists of DTS ports that implement communications ports tailored for

command/response and data.

3.1.2 Server Profile

The server profile implements the OTA upgrade message command set and data transfer protocols. The server leverages the data transfer service (DTS) ports and data flow behavior and requires the following port types:

- 1 bulk port (optional fast write data transfers) - 512 byte wide
- 1 mailbox port (control/status/data) - 512 bytes wide

The default service UUID is: a0e78d39-75b5-4182-8fdc-c4b7365c9062
Customers can reconfigure the profile to install their own UUID.

Table 1 - Attribute Definition

Attribute	Attribute Type	Permission	Values	Notes
Service	Primary Service Declaration	RO	Service UUID defined by the user	No include declaration needed. UUID: default ATM OTA
Bulk Port Char	Characteristic Declaration	RO	Properties, Value Handle, UUID	UUID: 040C6507-F08A-4FC2-ABEA-6382EE250230
Bulk Port Char Value	Characteristic Value Declaration	Write No-RSP / Notify	512 -byte value size	Bulk Data Flow (Stream). Max transfer ATT_MTU - 3
Bulk Port CCC	Client Characteristic Configuration Descriptor	R/W	CCC bit	Set value to 0x0001 to enable ATT notifications
MBOX Port Char	Characteristic Declaration	RO	Properties, Value Handle, UUID	UUID: CB91B0D6-3080-4DFB-876B-407DB045E52B
MBOX Port Char Value	Characteristic Value Declaration	Write / Indicate	512 -byte value size	Mailbox Data Flow. Max read/write of 512 bytes. Max indicate is ATT_MTU - 3
MBOX Port CCC	Client Characteristic Configuration Descriptor	R/W	CCC bit	Set value to 0x0002 to enable ATT indications

3.1.3 Upgrade Processor

The upgrade processor is a reusable library that implements the image upgrade command set. There is only one instance of the upgrade processor in the system and it will be initialized and managed by the server profile. The Bluetooth LE server profile

provides an over-the-air messaging channel that bridges the client to the upgrade processor. For most commands the Bluetooth LE client is interacting mainly with the upgrade processor. The upgrade engine interfaces with the RRAM subsystem to access and transfer data to RRAM partitions.

3.1.4 Storage Subsystem

This consists of the storage subsystem (drivers) that manage the underlying storage medium. The upgrade processor uses an upgrade storage interface to support internal RRAM and in the future flash storage to allow the implementation of a hybrid upgrade scheme.

3.1.5 RRAM

The RRAM subsystem is divided into areas containing code and data as shown in Figure 2:

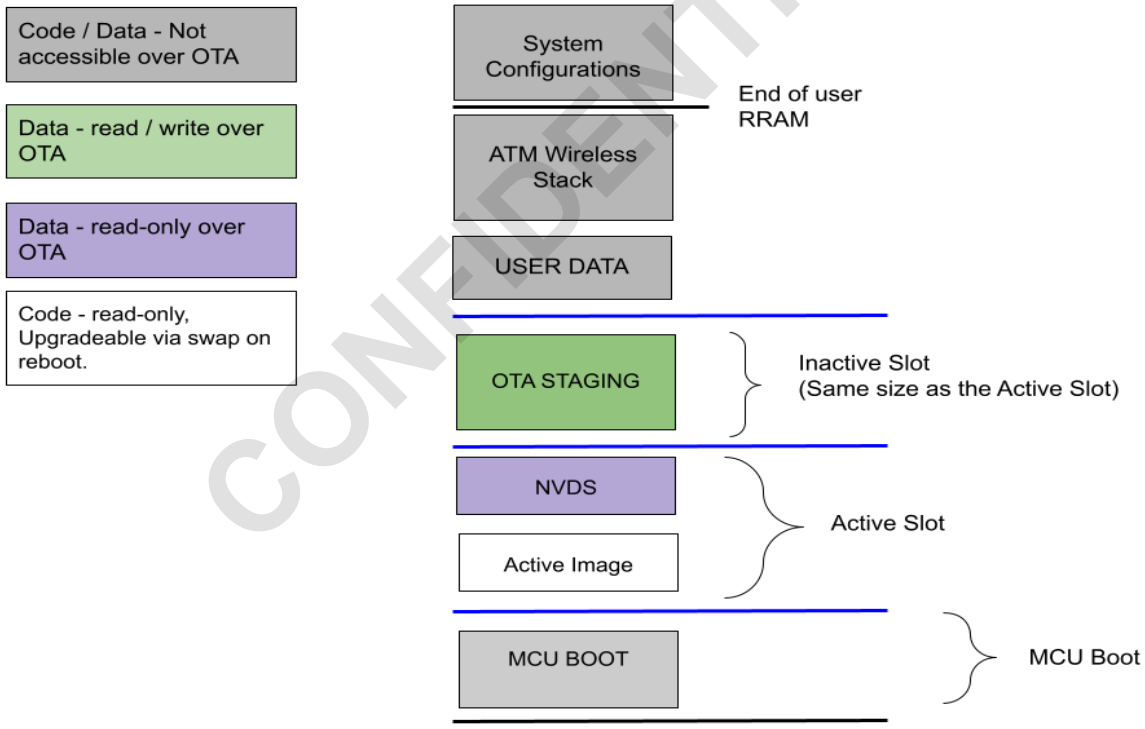


Figure 2 -RRAM Subsystem Block Diagram

The OTA server allows access to the OTA staging area (read/write), NVDS (read only) and the active image area (read only). Details of each area is described in the SDK Users Reference Manual. In previous external flash-based products, the partitions were binary (half the storage device size) with sub-partitions laid out for NVDS and user data.

For ATM33, the partitions are segmented to support reliable image upgrades with provisions for user data and NVDS. In ATM33 the upgrade partition term is synonymous with MCUBoot's slot nomenclature. In previous generations the OTA server protocol referred to these as active/inactive upgrade partitions. For further discussions, an active slot is equivalent to an active partition and the inactive slot is equivalent to the inactive partition.

3.1.6 MCUBoot

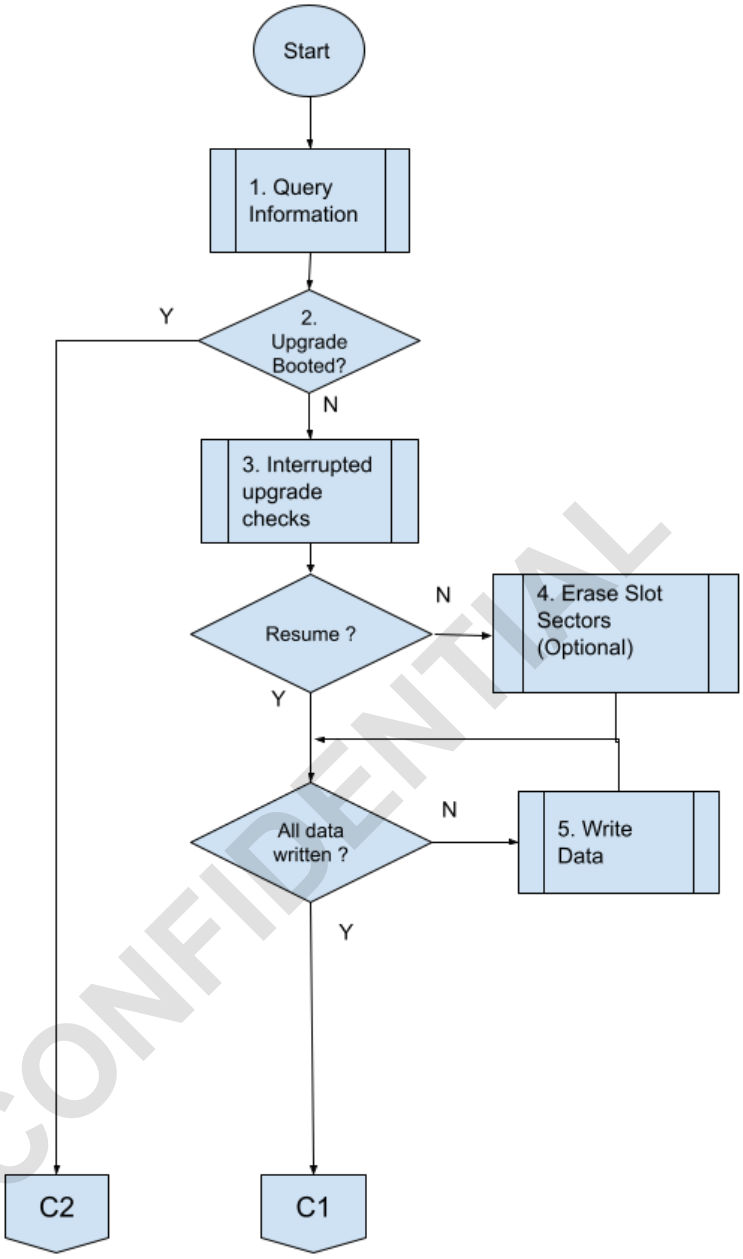
The system boot chain begins with the ROM and transfers control to a primary bootloader stage in RRAM. The current primary bootloader is based on the open source MCUboot project. This bootloader is non-upgradeable and is only used to validate the currently active image for integrity and optionally authenticate the image (secure boot). The primary bootloader is also tasked to detect an image update request and swap images between the active and inactive slots. A recovery mechanism is employed that detects a failed boot or an unacknowledged boot (first boot test that fails to be acknowledged) and will automatically swap the original image back.

3.1.7 Upgrade State

The server profile does not maintain the state of the upgrade process. Upgrade session state is not maintained between re-connections or power/reset cycles. Any upgrade state can be determined through an inspection of the inactive slot's stored data. The OTA profiles provide methods to aid in assessing the programmed state of the inactive slot.

3.2 Upgrade Process Flow

The high level image upgrade flow is described in the following diagrams. **Note**: Blocks that are enumerated (1.,2.) have a corresponding detailed description in the Upgrade Flow Step Details section.



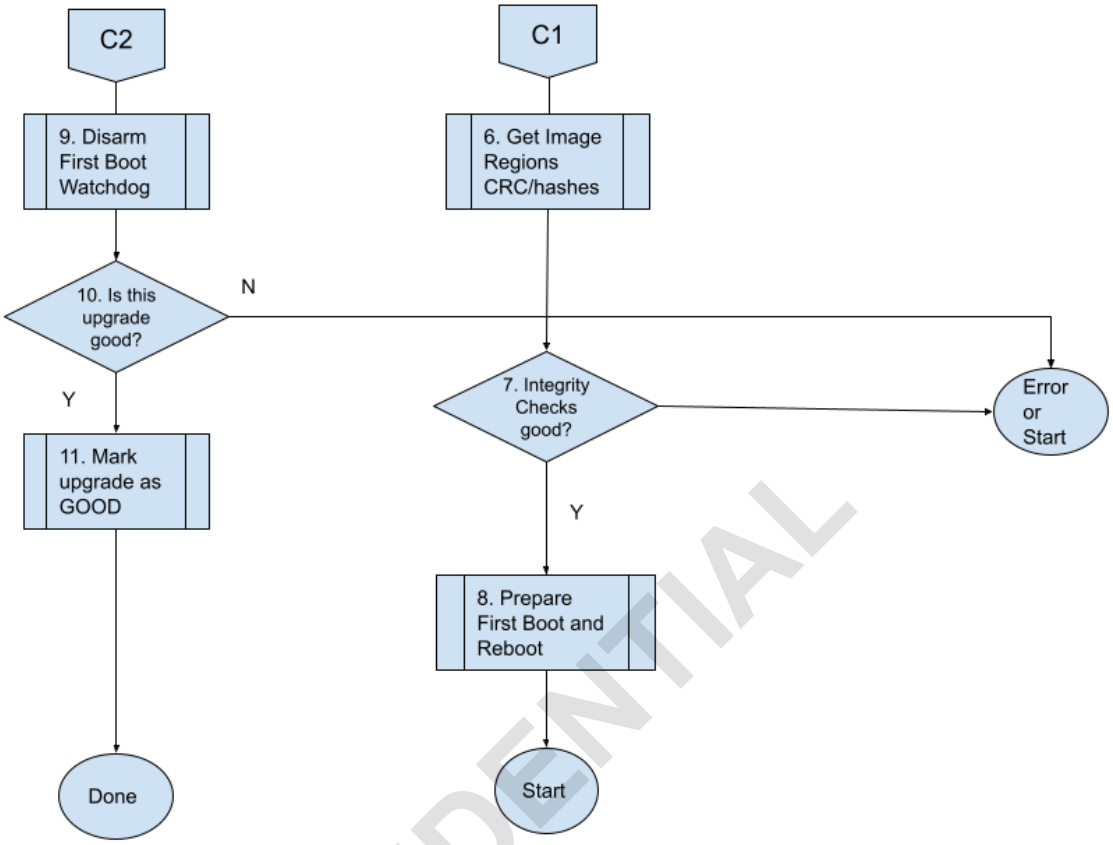


Figure 3 - Image Upgrade Flow Diagram

3.2.1 Use of a CRC/Hash

The upgrade process makes heavy use of CRC codes to verify the integrity of the data written to RRAM. This avoids costly data reads to verify programming. Data in transit however is not CRC/hashed as the transport can provide data integrity checks in transit. Ultimately the determinant of programming success and failure rests on what is actually programmed. The profile provides commands to CRC/hash multiple regions disjointly with one command. The CRC selected is CRC-32 for implementation simplicity, relative speed and error detection.

3.2.2 Upgrade Flow Step Details

The enumerated steps in the previous diagrams are listed below.

1) Client connects to the server profile and queries information regarding the platform and upgrade state of the system.

2) The state of the active slot is checked to determine if an upgrade is in first boot. If the active slot indicates it is a test boot, then the inactive slot contains the recovery image. The upgrade processor will disable all writes to the inactive slot

in this case. The client is not allowed to update the contents of the inactive slot during a first boot. The client can only mark the first boot as good.

3) In an interrupted upgrade, the client determines where to resume writing data in the inactive slot. If the client wishes to optimize this process they have the option to employ their own algorithms to determine where the write progress left off. This is done with a combination of CRC commands "walk" the inactive partition (slot). The client can get a reasonable assessment of where it left off. Otherwise the client can just start fresh and begin programming the slot.

4) For RRAM storage this is optional. The upgrade processor will report that erase is optional for the underlying memory technology. The OTA protocol supports erase semantics if the client chooses to erase the contents (written to 0xFFs) in the inactive slot.

5) Data blocks of the image are written to staging buffers and then buffers are written to RRAM. If a write operation is interrupted (disconnect or system reset), the client is required to determine where it left off when it reconnects and should make no assumptions about whether the last operation succeeded or not.

6) The client should request CRC/hash values for memory regions that fully validate the upgrade. It is recommended that the regions should cover 4-32K blocks to help debug a cell programming problem.

7) If the integrity check fails the client can retry if required.

8) At this step the client has determined that the slot has been properly programmed. It is now ready to finalize steps to test boot the new upgraded partition. At this step the inactive slot is ready for a test boot and is not considered GOOD until the client has had a chance to reconnect to the image on first boot. The client requests a test boot which will flag, in a non-volatile manner, MCUboot to perform the image swap at the next boot. The system will disconnect and reboot back to MCUBoot. **Note**: The image swap operation can take some time, the client should continue to scan for the device. The amount of time it takes to swap and reboot depends on the size of the upgrade. On a typical image this can range from 30 seconds or more.

9) A new upgrade has booted through the mechanism in step 8. In the event the client cannot connect back to the OTA server there is an optional first boot watchdog mechanism that will trigger a reset to return back to MCUBoot which will swap the last known good image. The first boot watchdog is disabled after the client can successfully query information from the upgrade processor over

BLE. Any reboot that occurs while the image is still marked as a test image will result in a reversion back to the last known good image.

10) The client now determines whether the upgraded image is running properly. Note that the upgraded image has been swapped and is now running in the active slot. The upgraded image sees the same NVDS. NVDS is currently not swapped in this version.

11) This step should ONLY be done when the upgraded image has been swapped to the active slot through a boot back through MCUBoot (step 8). If the active slot meets the client's criteria for a good boot, then the client can now mark the image as GOOD. This is irreversible and unlocks the inactive partition for future upgrades. A command is issued to mark the current active slot (first booted) as GOOD. Any reboots after this point will not revert the image.

3.2.3 First Boot Watchdog

The server profile can be configured with a first boot watchdog. This watchdog can be enabled if the active slot is detected as a first boot. The client has until the watchdog timeout to connect and perform a query information otherwise the watchdog will reboot the platform and MCUBoot will swap back to the last known GOOD image. The watchdog timeout is typically 30 seconds.

3.3 Messages

3.3.1 Types

The upgrade protocol uses the following 4 types of messages:

Command

A control message with variable length parameters. Length of parameters depends on the message. A command always results in a status message with an optional completion message for long running commands.

Status

Immediate status reporting of the last command. The status message may also contain a variable number of parameters which are specific to the command. For example read data or area CRC results.

Completion

Completion messages indicate the completion of long running commands (for example: erase) that report final status separately from a status message. Whereas the command status indicates a command is accepted (valid parameters) with PENDING the completion message reports the final result of the operation.

Write Data

Write messages follow the same format as commands however their delivery and status follow a slightly different flow. Write data messages are fragmented messages to post data to a server-side staging buffer. One or more write data messages will have a single corresponding status anda final completion response . A status is returned on bad parameters, missing fragment (after some timeout) or on a successful transfer right before the programming operation is started. A PENDING status is an indication that the client can now move more data to another staging buffer while programming is in progress. The client can also expect a completion event when programming is complete (either success or failure).

3.3.2 Ports

The following section describes the two port types used for all communications with the server profile. These ports are defined by the Atmosic Data Transfer Service and also used in the Throughput testing profile.

Mailbox

The mailbox is reserved for all command, status and completion messages. Transfers to/from the mailbox use the write-with-response and indicate attribute access procedures. This form of transfer, although reliable and avoids potential resource contention issues, is intended for infrequent message exchanges (i.e. control/status). The control protocol is designed such that most messages can fit into a single ATT transfer procedure (single ATT PDU with confirmation down to an MTU size of 23 bytes) however some messages may exceed the indication size limit (ATT_MTU - 3) and require a client initiated read long ATT procedure. Write data transfers to the write buffers can also be performed through the mailbox. This can simplify implementations that do not choose to implement the retransmit logic required over the BULK port.

Bulk

The bulk port provides the fastest and most efficient way to transfer data to waiting write buffers. Data message fragments are sequentially streamed into the write buffers, however, some fragments could be dropped (considered uncommon in most cases). To

maximize the performance through the bulk interface, programming and staging buffer transfer should be overlapped when possible.

3.3.3 Definitions

Message Format

All fields follow little-endian ordering. For Bluetooth LE, the maximum length of a message is 512 bytes.

Table 2 – Message Format

Size	Field	Description
1	**Msg ID**	**Message ID is 1 octet (1-255), ID 0 is reserved**
n	<parameters>	Variable length parameters (See Message size limits)

All messages are encapsulated into the transport's data framing mechanism and thus a length field is not required. The transport layer conveys the length of the full message after decapsulating it and removing any padding. Transport layer is expected to deliver single unfragmented messages.

Table 3 – Status Codes

Name	Value	Description
SUCCESS	0	Success
PENDING	1	Success but the operation is pending and will complete with a completion message
INV_PARAM	2	One or more parameters of the command are invalid
UNSUPPORTED	3	Unsupported command or operation
WRITE_FAILED	4	Failure of a write operation. This status is also used by operations that perform data checks on the integrity of the write (see Copy and Set Upgrade Status).
ERASE_FAILED	5	Erase operation failed.
BUSY	6	Operation could not be completed because the resource is busy.
MISSING_FRAG	7	One or more fragments are missing in the data transfer request

NOT_ERASED	8	The erase operations's post verify operation detected a non-erased byte or a command could not execute because a location was not erased.
INV_PART_STATUS	9	The server encountered an invalid partition status for the requested operation (see Set Upgrade Status)
WRITE_BUF_OVERFLOW	10	The last write data command has overflowed the write buffer
RESP_OVERFLOW	11	The resulting status or completion response overflows the maximum length allowed by the transport. This can happen if the client requests more read data than the transport can deliver in one response. See Message size limits
CANCELED	12	Operation was canceled
RD_RAILED	13	Read operation failed
INVALID_REQ	14	Invalid Request

Command Status

A command status message conveys the success or failure of the last command message. For a command that may require a completion, a PENDING status message indicates the operation has been accepted (parameters validated) and started. Some command processing ends with a status while others may require both a status followed by a separate completion message (i.e. a long running operation completes).

Size	Field	Description
1	**Msg ID**	**0x01**
1	Cmd Msg ID	The message ID of the command this status is for
1	Status	Command processing status code
N	<parameters>	Variable length status parameters (See Message size limits)

Command Completion

A command completion message conveys the success or failure of an operation that requires a delayed indication (long running operation). A long running command returns STATUS_PENDING in the status message and provides a final completion status in a completion message shown below:

Size	Field	Description
1	**Msg ID**	**0x02**
1	Cmd Msg ID	The message ID of the command.
1	Status	Final status of the completed long-running operation
N	<parameters>	Variable length status parameters (See Message size limits)

Cancel

This command cancels the current pending operation.

Size	Field	Description
1	**Msg ID**	**0x0F**

This message returns a status message with no parameters. The PENDING status will indicate that the cancellation is pending. An operation that was cancelled indicates a completion status of CANCELLED. If there is no pending operation at the time the command is received the status returns SUCCESS and no command completion will be issued

Bluetooth LE Considerations

Message size limits

For BLE GATT transports, the message size and parameter length limits are listed below:

Message Type	Mail Port (Max. Total Length / Max. Parameters Length)	Bulk Port (Total Length / Parameters Length)
Command	512/511	N/A (not exchanged on this port)
Status	512/509	N/A
Completion	512/509	N/A

Read	See status message, the maximum read length is limited by the maximum parameter length.	N/A
Write Data State	Same as command	ATT_MTU[1] - 3 / ATT_MTU - 4
Write Data Continuation	Same as command	ATT_MTU - 3 / ATT_MTU - 4

Pending Message Indication

For BLE GATT implementations any message that exceeds the size allowed for an indication (ATT_MTU-3) requires the client to read the message from the port using a read long attribute procedure. To indicate that the client must use this procedure the following message is defined.

Size	Field	Description
1	**Msg ID**	**0xF0**
1	length	The length of the pending message that requires a read long procedure.

This is typically used when the command status or completion exceeds the length that is allowed in an attribute indication.

Locking and Unlocking Sessions

Bluetooth LE applications can enter operating states that can interrupt OTA updates. For example, a device may disconnect all links in preparation for low power mode. Devices in low power modes may require external stimulus or user intervention to resume normal operations where connections are permitted. Applications can implement out-of-band techniques (switches, buttons) that place devices in OTA friendly states to ensure a timely update and thus improves the user experience. In the absence of such techniques an over-the-air signaling method can be used to lock a connection that is hosting an OTA session in order to prevent the device from entering states that disrupt the upgrade process. The LOCK/UNLOCK session message signals the client's

[1] ATT_MTU - Attribute Maximum Transmission Unit (see Bluetooth Specification)

intent to perform an uninterrupted OTA upgrade. This message is propagated as a software notification to the application. The details of how the application processes the locked/unlocked state is beyond the scope of this document. In general the locked state of the session will prevent the system from entering a state that interrupts the update process. The client can unlock the session and allow the system to return to normal operation upon completing upgrade tasks. Additionally a disconnect as a result of the client or the environment automatically unlocks the session.

The client can signal locking and unlocking a session using the following messages :

Lock Session

The client can lock the current session to avoid interruptions to the upgrade process. This command returns a status message with no parameters. If the session is already locked the message will fail with INVALID_REQ.

Size	Field	Description
1	Msg ID	0x12

Unlock Session

The client can unlock the current session and return the system to normal operation. This command returns a status message with no parameters. If the session is not locked the message will fail with INVALID_REQ. A session is automatically unlocked if the session's connection terminates while it is locked.

Size	Field	Description
1	Msg ID	0x13

Query Information

The client can query the properties of the upgrade server with this command.

Size	Field	Description
1	Msg ID	0x10

This command returns a status message with the following parameters:

STATUS Parameters		
Size	**Field**	**Description**
1	version	Upgrade protocol version information and flags: Bits : 0..3 : Protocol version 0x0 - v1.0 0x1 - v1.1 0x2 - v1.2 0x3 - v1.3 (This version) Bits : 4..7 : RFU
1	capabilities	Capabilities flags Bit : 0 : Stream Mode 1 - Supports write buffer stream protocol[1] 0 - not supported Bit : 1 : Dual write buffer support 1 - Supports dual (ping-pong) buffers[1] 0 - Single buffer Bit : 2 : Resizeable write buffers 1 - write buffers can be resized 0 - write buffers are fixed Bit : 3 : Memory Model 1 - small memory model - this affects the units of measure for the partition size 0 - normal memory model (default) Bit : 4 : Optional Erase 1 - Erase is not required to program 0 - Erase is required to program (default) Bit: 5 : Upgrade Type 1 - MCUBoot (ATM33 and later) 0 - Legacy (ATM2) Bits 6..7 : RFU 1. For BLE-GATT Profile : • If the profile does not support the data stream protocol the client must use the simpler (slower) method over the Mailbox. • If the profile supports dual buffers, data transfers can overlap with programming otherwise the client must serialize to a single buffer. (**As of this time writing dual buffer support is not supported**)
2	boot_id	Vendor specific board identifier - only understood by the client
1	buffer_size	Write buffer size is 256 byte pages (8 = 4 KB)
1	eraze_size	Erase sector size in 256 byte pages (16 = 4 kB)
1	flash_type	Flash type

2	capacity	Storage capacity in 16 KB units (max 1 GB). The capacity is only for informational purposes
2	part_size	Partition (slot) size in 16 KB units. If the capabilities flag indicates a small memory model this field represents 1 KB units
3	NVDS_data_offset	This is the upper 3 bytes of the offset of the NVDS for flash devices. The offset should be left shift by 8 before using it in commands (read or copy).
1	NVDS_size	The size of the NVDS in 1 KB units.
1	active_status	Bits 0..3 Status The following enumerations indicate the partition status 0x0 : reserved 0x1 : The partition is not fully programmed[2] 0x2 : The partition has been programmed[2] 0x3: The partition is a first boot[3] 0x4 : The partition is good 0xF : The partition has been marked bad Bit 4..7 RFU
1	inactive_status	The upgrade status of the inactive partition or slot (see active_status)
4	debug	Debug information (internal only)
2	mtu	Transport MTU size (BLE link MTU size for BLE transports)

Query Version Information

The client can query the properties of the device. If the device does not include the Device Information Service the client can use this method as an alternative.

Size	Field	Description
1	Msg ID	0x11

This command returns a status message with the version information set by the device. The maximum version length is 255 bytes. The actual version information is limited by the transport limits for a status message (See Message size limits).

[2] This status is not used when the upgrade type is MCUBoot

[3] The first boot status supplies only to the active partition.

STATUS Parameters		
Size	**Field**	**Description**
N	Version Information	Version information. A string or data structure defined by the versioning method. If a string is provided, the buffer may or may not contain a NULL terminator. If the device has not set the version information , this field will be empty.

Write Data Start

The client sends data to the write (staging) buffers as a sequence of messages. The first message is a start message followed by one or more continuation messages. The last fragment in the continuation starts programming. A start message marked as the only fragment also triggers the programming operation. Writes can only be issued for the inactive partition(slot). Write operations cannot be issued when there are outstanding copy or erase operations. Writes assume the sectors have been erased prior.

Size	Field	Description
1	**Msg ID**	**0x20**
1	flags	Write flags Bit 0 : Buffer selection (if dual buffer support) 0 - Fill Buffer 0 (default) 1 - Fill Buffer 1 Bit 2..7 : RFU (set to zeros)
4	offset	Starting offset into the inactive partition (slot) where the write buffer will be written to.
2	length	Length of all bytes to be written to the partition
1	count	Count of fragments (1-255) to send (this message is fragment 0). A count value of 0 indicates this buffer contains all the data for the operation and is considered the last fragment.
1	req_seq	Client specific, unique, per-request sequence number (returned in status and completion messages) to track the overall write operation.
<N>	data	Write data up to the remaining length of the first fragment

A status message is issued on the reception of all or missing (a timeout) sequences with parameters shown in the following table.

STATUS Parameters		
Size	**Field**	**Description**
1	req_seq	Request sequence copied from the start message
1	seq_good	The highest in-order received sequence number. Can be checked if status is MISSING_FRAG. This is only used if retransmit is supported.

If the status reported is PENDING the parameters/data are accepted and the programming operation has commenced. All other statuses are considered errors and no completion message is expected. The client can wait for a completion message or queue data to the other buffer (if dual buffer mode is supported, **at this time dual buffer mode is not supported)**. Any transfer to a buffer that is being programmed will be rejected with an error.

In stream mode, all reception status indications are deferred. This allows the client to stream the start followed by continuation messages without pausing for status messages. Since it is possible that any message or even all messages including the start message could be lost the status check at the end of the sequence must be checked with a timeout.

The following table shows the parameters for the completion message:

COMPLETION Parameters		
Size	**Field**	**Description**
1	req_seq	Request sequence from the start message

Write Data Continuation

This is a continuation message to deliver a write data fragment. The write buffer offset is required since the start message also delivers data into the buffer. The next continuation message should start where the previous message left off. Also in the event a fragment is re-sent the buffer offset ensures that the data is filled at the correct location in the write buffer. See Message size limits for the upper limit of a fragment.

Size	Field	Description
1	**Msg ID**	**0x21**
1	req_seq	Sequence number for this request. Must match the write start message.
1	seq	Fragment sequence number (1-255). The Write Start message is always sequence 0.
2	buff_offset	Starting offset into the write buffer for the following data
<N>	data	Data up to the remaining length of this fragment

This message uses the same status parameters as Write Data Start.

In stream mode, all reception status indications are deferred. This allows the client to stream continuation messages without pausing for status messages. A final check of status with a timeout is required at the end of the sequence.

Copy Data

This command can copy a single region from the active partition to the inactive partition/slot. This command is useful for copying NVDS data to the inactive partition/slot without having the client read out the data. If erase is not optional, the inactive partition/slot is assumed to be erased.

Size	Field	Description
1	**Msg ID**	**0x22**
1	flags	Copy flags Bit 0 : Copy check 0 : no copy checks during the copy 1 : perform data checks during copy Bit 1..7 : RFU (set to zeros)
4	dest_offset	The destination offset in the inactive partition to copy to
4	src_offset	The source offset in the active partition to copy from
4	length	Length of data to copy.

The status message returns a status of PENDING if the parameters are accepted and the copy operation has started. A completion message with the final results of the copy is provided. A failed copy check returns a status of WR_FAILED in the completion.

Last Write Status

This message retrieves the last write status. This status can be used to determine the cause of a status timeout during a stream mode write buffer sequence.

Size	Field	Description
1	Msg ID	0x23

The message returns a status message with the following parameters:

STATUS Parameters		
Size	**Field**	**Description**
1	req_seq	Request sequence tag copied from the start message. In stream mode if the request sequence reported here is different than the expected request sequence then the write start message was likely dropped
1	seq_good	The highest in-order received sequence number from the last req_seq tagged sequence. In stream mode this value can determine if any intermediate fragments (WR CONT messages) were dropped
1	t_count	Fragment count. In stream mode this is only valid if the start message was received.

Read Data

The client can read data in either partition. This command is not optimized to read large amounts of data and intended for small extractions useful in debug or inspection (ex: inspecting NVDS).

Size	Field	Description
1	Msg ID	0x30
1	flags	Read flags Bit 0 : Partition selection 0 - Inactive Partition or Slot (default) 1 - Active Partition or Slot Bits 1..7 : RFU (set to zeros)
4	offset	Offset in the partition to read
2	length	Length of data to read (see Message size limits)

The message returns a status message with the following parameters:

STATUS Parameters		
Size	**Field**	**Description**
N	Data	Data read from the offset to (offset + length) (see Message size limits)

Get Programmed State

The client can obtain CRC or erase state information across multiple block ranges.

Size	Field	Description
1	**Msg ID**	**0x31**
1	flags	CRC/ Erase Verify flags Bit 0 : Partition selection 0 - Inactive Partition (default) 1 - Active Partition Bit 1 : Mode 0 - CRC calculation 1 - Erase state Bits 2..7 : RFU (set to zeros)
N	range_desc	Range descriptors repeated <N> times (see below)

Range Descriptor		
Size	**Field**	**Description**
4	Offset	Offset in partition
4	Length	Length of data to check (32 KB max for CRC)

For Bluetooth LE the maximum number of range entries is limited by the maximum command parameter size (See Message size limits).

This command returns a status of PENDING if parameters are accepted and the operation has been scheduled. A final command completion message is issued with the following parameters:

STATUS Parameters		
Size	**Field**	**Description**
4*N	CRC or erase state for range repeated N times	If CRC : Bits 0..31 : The 4-byte CRC value If erase state: Bits 0..1 : Erase state : 0 = reserved 1 = erased 2 = not erased 3 = reserved Bits 2..31 : RFU (See Message size limits)

Erase Sectors

The client can erase sectors in one or more regions in the inactive partition/slot. The client can optionally request an erase verification step that can report whether any region contains a non-erased byte. This is reflected in the command completion status as NOT_ERASED. Erase operations cannot be issued if there are pending write/erase/copy operations. Erase is not required if erase is optional. For RRAM based systems this command will fill the sector with 0xFFs emulating a flash erase.

Size	**Field**	**Description**
1	**Msg ID**	**0x31**
1	flags	Erase flags Bit 0 : Erase Verify 0 - Do not perform erase verify (default) 1 - Perform erase verify Bit 1..7 : RFU (set to zeros)
N	range_desc	Sector descriptor repeated <N> times (see below)

This command returns a status message with no parameters. If the status is PENDING, the erase parameters have been accepted and the erase operation is pending. Erase operations cannot be overlapped.

Set Upgrade Status

This command sets the upgrade status for the inactive or active partition/slot. The “PROGRAMMED” status is applied only to the **inactive** partition in order to allow a first-boot restart where the newly programmed partition is swapped as the **active** partition. The “GOOD” status only applies to the active partition that is operating in first boot mode.

Marking an active partition that is already “GOOD” returns an error. The client must determine whether the new image operating in first boot mode qualifies as a good image and then and only then set this status. An **inactive** partition CANNOT be marked GOOD. The BAD status can only be applied to an **inactive** partition. In order to set a BAD status the client must reboot the system into the last known GOOD partition and mark the **inactive** partition. This status can be used to quickly mark an upgraded image in the inactive partition that has not properly booted into the first boot stage. PROGRAMMED and BAD statuses only apply to non-MCUBoot upgrade types. Setting these statuses on MCUBoot upgrade types is ignored. It does not affect how MCUboot swaps images.

Size	Field	Description
1	**Msg ID**	**0x50**
1	status	Partition/Slot status to set 0x0 - Reserved 0x1 - Reserved 0x2 - Set the ***inactive*** partition as PROGRAMMED[4] 0x3 - Set the ***active*** partition as GOOD 0xF - Set the ***inactive*** partition as BAD[2]

This message returns a status message with no parameters. The status will indicate whether the desired status could be applied to the targeted partition.

Reboot

This command allows the client to remotely reboot the system to revert to the last known GOOD partition or validate the first boot of a new upgrade

Size	Field	Description
1	**Msg ID**	**0x51**
1	type	Reboot type 0 - Normal reboot 1 - Perform first boot[5] of the inactive partition/slot

[4] No-opts when upgrade type is MCUBoot.

[5] For non-MCUboot types, the first boot reboot type should only be requested if the inactive partition has been marked as PROGRAMMED (see Set Upgrade Status).

The reboot returns a status with no additional parameters, however the status may not arrive before the system reboots.

3.4 Message Flows

The following section provides flow diagrams for commands, status, completions and data. See Figure 4.

3.4.1 Control

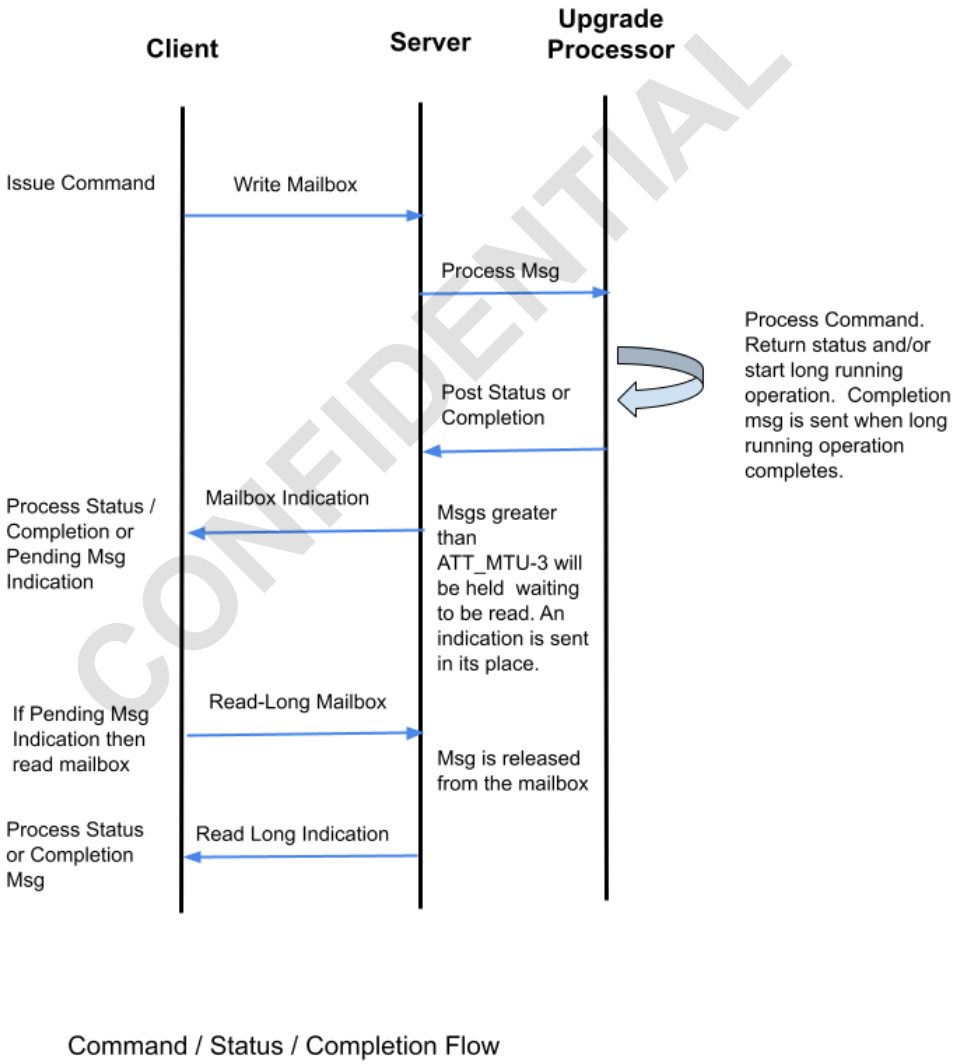


Figure 4 - Command/Status/Completion Flow Diagram

3.4.2 Write Data

This is the default flow that uses the mailbox port to transfer data to the write buffers. All start and continuation messages report status as mailbox indications. The client must wait for the status before issuing additional messages. This increases overhead but provides reliable and easily debuggable message sequencing. See Figure 5.

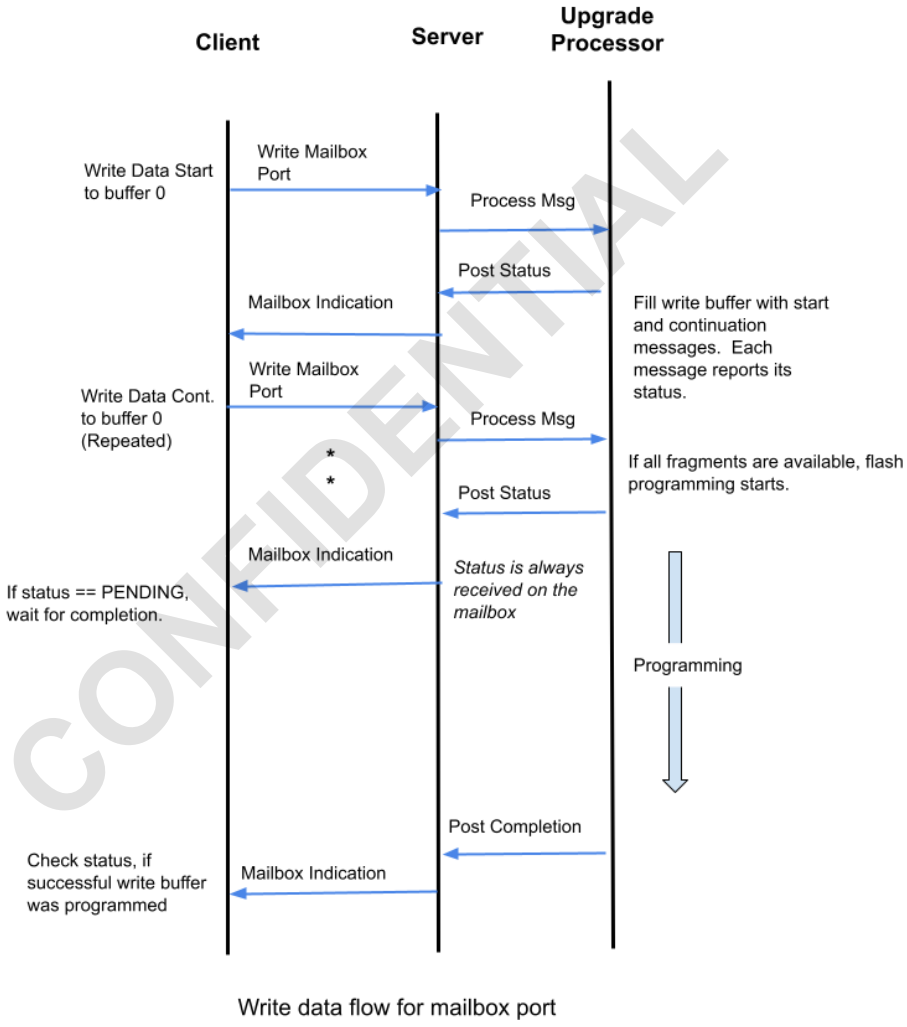


Figure 5 - Write Data Flow for Mailbox Port

3.4.3 Write Data Stream Mode

This flow uses the bulk port to stream write start and write continuation messages to fill the write buffer. This flow uses write-with-no-response GATT procedures and the

payload of messages is limited by the negotiated MTU size in addition to the characteristic size . In this flow all intermediate status messages for each start and continuation message regardless of success or failure are omitted with the exception of the final message (start or continuation) that results in programming (i.e. the buffer is filled and ready for programming). In this mode the entire sequence of messages is considered one request and a final status is reported. The status is always reported on the mailbox port. In some client platforms and under certain conditions, the BLE stack may optionally drop GATT write-no-response messages that do not require confirmation. This introduces the possibility of one or more (and possibly all) messages in a sequence to be dropped. This requires the client application to employ a timeout when expecting a status at the end of a buffer fill sequence. A client can expect a final status that indicates programming has begun or receive no status at all (timeout) indicating that one or more messages in the sequence was dropped including the possibility of dropping the start message. This is illustrated in Figure 6:

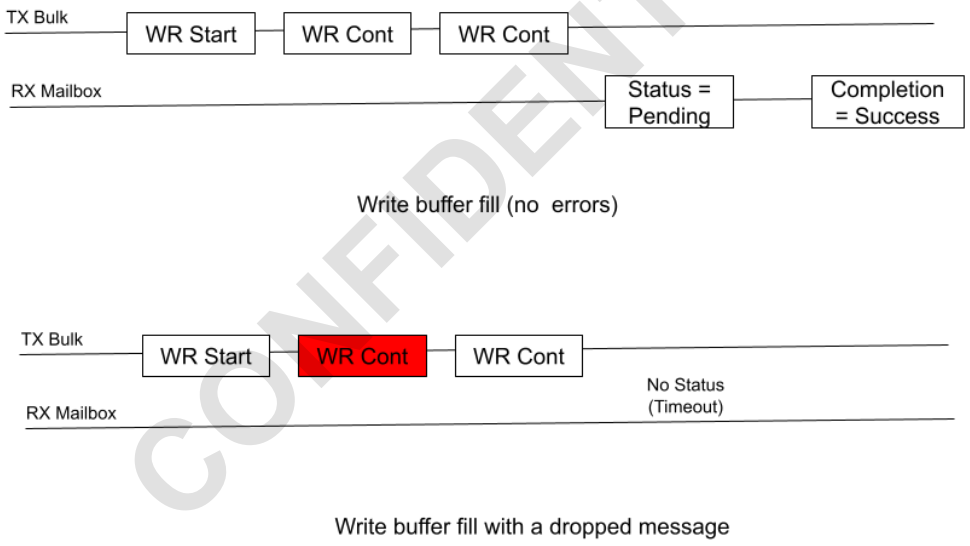


Figure 6 - Write Buffer Fill with a Dropped Message

The use of a timeout is required in the event the entire sequence is dropped. The timeout should be selected based on the number of messages (start + continuations) multiplied by the supervision timeout. For example, given a typical 100ms link supervision timeout and 5 messages (1 start and 4 fragments), the timeout for the status should be set to at least 500 ms.

In the event of a timeout the application can retry the buffer write sequence or alternatively switch to writing via the mailbox. The “Last Write Status” command can be issued to get the last write stream status to determine the cause of the timeout.

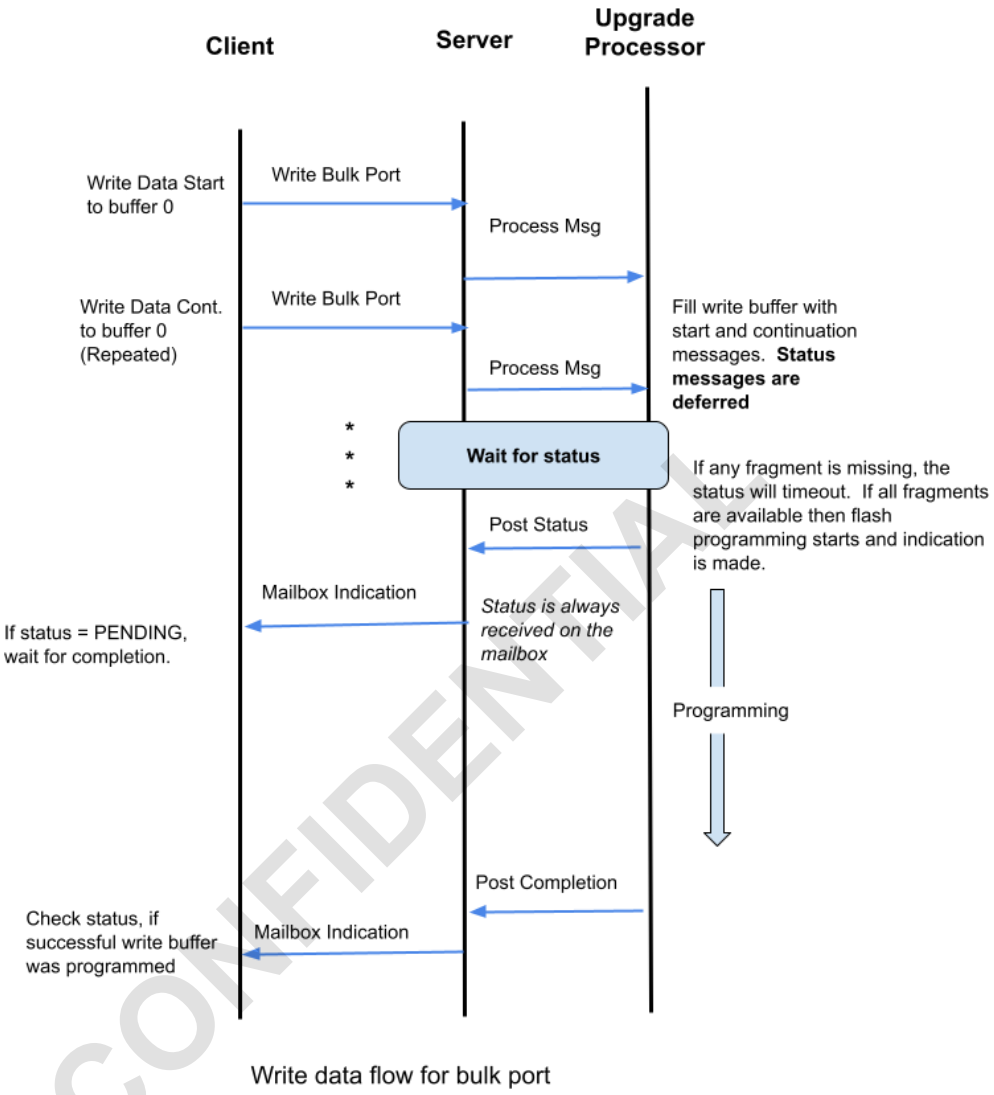


Figure 7 - Write Data Flow for Bulk Port

4. Developers Guide

4.1 Installing the profile

The following code snippets can be found in the TPUTP_server example. The following code shows the installation of the profile with default example settings using the ble_otaps library module :

```
// Enable the OTA Service

static struct otaps_cfg const ota_config = {
   .wr_buffer_size = ATM_TPUTP_WR_BUFFER,
   .board_id = ATM_TPUTP_BOARD_ID,
   .fboot_timeout = ATM_TPUTP_FIRST_BOOT_TIME,
   .ver_info = APP_DIS_FIRM_REV_STR,
   .ver_length = APP_DIS_FIRM_REV_STR_LEN,
};
static struct otaps_param const otaps_parm = {
   .sec_lvl = OTA_NO_AUTH,
   .cfg = &ota_config,
   .ota_lock_ind = server_ota_lock_ind,
};
atm_gap_prf_reg(BLE_OTAPS_MODULE_NAME, &otaps_parm);
```

The OTA configuration data is defined in the otaps_cfg structure. This structure is passed to the profile via `atm_gap_prf_reg()`.

4.1.1 Board Identifier

The board identifier is specific to the customer's design and only understood by the client profile. This value can be hard coded by the application or read from tags stored in system configuration data.

4.1.2 Write Buffer Size

This value indicates the maximum size of the write buffer to allocate. The recommended value is at least 256 bytes. The application must determine a suitable value given its memory constraints.

4.1.3 First boot timeout

If this value is non-zero, the profile will start a first boot watchdog when the active partition's first boot is detected. The time value is defined in scheduling time units (10 ms). The client must connect and send any message (i.e. Query Information) to stop the watchdog. Failure to do so will cause an automatic system reboot and the system will boot from the last known good partition.

4.1.4 Version String

The application can supply optional version information that can be passed to the OTA client. The example shows the version information set to a human readable version string. This option is provided for systems that may not implement the standard Bluetooth Device Information Service (DIS).

4.2 Changing the Service UUID

The UUID value is constant value that requires a modification to the profile source code. This is not run-time configurable. Locate the profile source module: `profiles/otap/otaps/otaps.c`.

The constant definition is shown below:

```
static const uint8_t otaps_uuid[ATT_UUID_128_LEN] = OTAP_SERVICE_UUID;
```

Where the array is defined below showing the current default value:

```
#define OTAP_SERVICE_UUID {0x62, 0x90, 0x5c, 0x36, 0xb7, 0xc4, 0xdc, 0x8f, 0x82,
0x41, 0xb5, 0x75, 0x39, 0x8d, 0xe7, 0xa0}
```

4.3 Configuring the Slot Sizes

Please refer to the SDK Users Reference Manual available on the Atmosic Support website on how to size and configure partitions and slots.

4.4 Cleaning Upgrades

The upgrade state is stored in the image trailer. When installing a new image via the debugger this area is cleared to a known state. This will prevent an inadvertent swap due to stale information in the image trailer.

4.5 Testing with a Non-Upgradeable Partition Scheme

During the early stages of development the application may be built with a non-upgradeable scheme to use a full debug version of the BLE stack. Although the application will contain a working OTA profile and upgrade processor, this build variant will disable upgrade support. You can still connect to the OTA server profile and query the device. The system will report the following console message in a debug build:

```
WARNING: MCUBOOT not configured, upgrades disabled
```

If an information query is performed the `debug` field in the query information parameters will contain the following token : `0xDEADBEEF`

In this mode, all operations that can mutate RRAM are disabled and will return an error This includes erase, write, copy and setting the upgrade state.

Revision History

Date	Version	Description
May 9, 2022	0.50	Initial version created

References

SDK User Reference Manual for ATM33xxe-3_ATM33xx-3

Glossary

Acronym	Description
GATI	Generic Attribute
DTS	Atmosic Data Transfer Service Characteristics
ATT	Attribute Protocol
MTU	Maximum Transmission Unit
BULK	DTS bulk data port
MBOX	DTS mailbox data port
OTA	Over the Air
RRAM	Resistive Random Access Memory

ATMOSIC TECHNOLOGIES – DISCLAIMER

This product document is intended to be a general informational aid and not a substitute for any literature or labeling accompanying your purchase of the Atmosic product. Atmosic reserves the right to amend its product literature at any time without notice and for any reason, including to improve product design or function. While Atmosic strives to make its documents accurate and current, Atmosic makes no warranty or representation that the information contained in this document is completely accurate, and Atmosic hereby disclaims (i) any and all liability for any errors or inaccuracies contained in any document or in any other product literature and any damages or lost profits resulting therefrom; (ii) any and all liability and responsibility for any action you take or fail to take based on the information contained in this document; and (iii) any and all implied warranties which may attach to this document, including warranties of fitness for particular purpose, non-infringement and merchantability. Consequently, you assume all risk in your use of this document, the Atmosic product, and in any action you take or fail to take based upon the information in this document. Any statements in this document in regard to the suitability of an Atmosic product for certain types of applications are based on Atmosic's general knowledge of typical requirements in generic applications and are not binding statements about the suitability of Atmosic products for any particular application. It is your responsibility as the customer to validate that a particular Atmosic product is suitable for use in a particular application. All content in this document is proprietary, copyrighted, and owned or licensed by Atmosic, and any unauthorized use of content or trademarks contained herein is strictly prohibited.

Atmosic™

Atmosic Technologies | 2105 S. Bascom Ave. | Campbell CA, 95008
www.atmosic.com