Small Satellite Research Laboratory
Franklin College of Arts and Sciences
UNIVERSITY OF GEORGIA

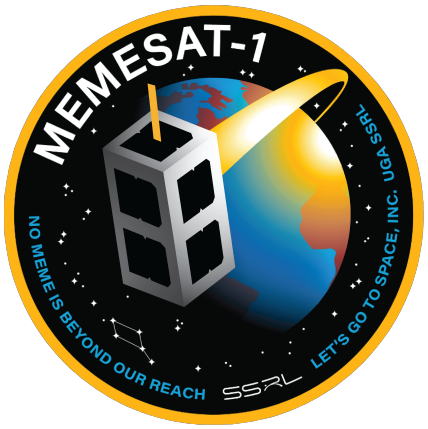
BBSMessageStorage Component Documentation

Prepared for
The Small Satellite Research Laboratory
Fall 2024 - Spring 2025

Principal Investigator
Dr. Deepak Mishra

The University of Georgia
Geography-Geology Department
Athens, Georgia
United States of America

Prepared by
The MEMESat-1 CDH Team
University of Georgia
Athens, Georgia
United States of America



Contents

1 List of Acronyms

BBS Bulletin Board System
CDH Command and Data Handling
EPS Electrical Power System
GDS Ground Data System
FSM Finite State Machine
FSW Flight Software
MCU Microcontroller Unit
MS-1 MEMESat-1
OBC On-Board Computer
OSAL Operating System Abstraction Layer
OTA Over-The-Air
OTAU Over-The-Air Updater
TLM Telemetry
UART Universal Asynchronous Receiver-Transmitter
UHF Ultra High Frequency

2 Revision History

Changes	**Authors**	**Version**
[2023-05-28] Wrote document as a documentation of the already finished source code	Marius Baden	1.0.0
[2024-03-08] Reviewing document contents for adjustment to account for current component functionality.	Samuel Lemus	1.0.1
[2025-04-01] Reviewing for IMR-3	Samuel Lemus	1.0.2

3 Summary

The MessageStorage component is a passive F' component responsible for storing and loading messages of type BBSMessage on/from the satellite's file system.

4 Design

4.1 Requirements

Requirement	Description	Verification Method
BBSMS_001	The component shall be able to store messages in a file system directory.	Unit Test
BBSMS_002	The component shall be able to load messages from a file system directory.	Unit Test
BBSMS_003	The component shall be able to prompt the deletion of files in a specified directory.	Unit Test

4.2 Custom Ports

Other components which want to use MessageStorage to store a message need to call its ports.

- storeMessage: Stores a single given message.
- loadMessageLastN: Loads a given number of the most recently stored messages. I.e., messages are handled in last-in-first-out order. The loaded messages are returned in a batch which is defined as a type.
- loadMessageFromIndex: Loads a single message from a provided index. The index is an identifier number internal to the component. This port is only useful if the user known what index they are looking for, e.g., from an event or telemetry data emitted by the component.

Name	Parameter	Type	Description
storeMessage	MessageSet	guarded input	Store a single given message at the next available index.
loadMessageFromIdex	MessageGetFromIndex	guarded input	Load a single stored message by index.
loadMessageLastN	MessageGetLastN	guarded input	Load the first n messages which have been stored the most recently and can be successfully loaded.

4.3 Custom Types

Name	Type	Parameter	Description
MessageWriteError	enum	FILE_EXISTS	A .bbsmsg file ith the specified index already exists.
		OPEN	File OSAL call to open the file failed.
		DELIMITER_WRITE	Writing the delimiter to the file failed.
		DELIMITER_SIZE	Writing the delimiter to the file did not write the expected number of bytes.
		MESSAGE_SIZE_WRITE	Writing the message size to the file failed.
		MESSAGE_SIZE_SIZE	Writing the message size to the file sys not write the expected number of bytes.
		MESSAGE_CONTENT_WRITE	Writing the message content to the file failed.
		MESSAGE_CONTENT_SIZE	Writing the message content to the file did not write the expected number of bytes.
		CLEANUP_DELETE	Deleting the file after an error occurred failed.

Name	Type	Parameter	Description
MessageReadError	enum	OPEN	File OSAL call to open the file for the specified index failed. Maybe the file does not exist.
		DELIMITER_READ	Reading the delimiter from the file failed.
		DELIMITER_SIZE	Reading the delimiter from the file did not read the expected number of bytes.
		DELIMITER_CONTENT	The delimiter read from the file does not match the expected delimiter.
		MESSAGE_SIZE_READ	Reading the message size from the file failed.
		MESSAGE_SIZE_SIZE	Reading the message size from the file did not read the expected number of bytes.
		MESSAGE_SIZE_ DESER_SET_LENGTH	Setting the length of the deserialization buffer for deserializing the message size failed.
		MESSAGE_SIZE_ DESER_EXECUTE	Deserializing the message size failed.
		MESSAGE_SIZE_ DESER_READ_LENGTH	Deserializing the message size did not use the expected number of byte.
		MESSAGE_SIZE_ EXCEEDS_BUFFER	The message size read from the file exceeds the size of the deserialization buffer.
		MESSAGE_SIZE_ ZERO	The message size read from the file is zero.
		MESSAGE_CONTENT_ READ	Reading the message from the file failed.
		MESSAGE_CONTENT_ SIZE	Reading the message from the file did not read the expected number of bytes.

Name	Type	Parameter	Description
MessageReadError	enum	MESSAGE_CONTENT_ DESER_SET_LENGTH	Setting the length of the deserialization buffer for deserializing the message content failed.
		MESSAGE_CONTENT_ DESER_EXECUTE	Deserializing the message content failed.
		MESSAGE_CONTENT_ DESER_READ_LENGTH	Deserializing the message content did not use the expected number of bytes.
		FILE_END	Parsing the message from the file ended before the end of the file was reached.

Name	Type	Parameter	Description
IndexRestoreError	enum	STORAGE_DIR_OPEN	Opening the storage directory failed.
		STORAGE_DIR_READ	Reading the file names from the storage directory ended with an error instead of OS::Directory::NO_MORE_FILES.

4.4 Events

The component emits an event every time

- It successfully stores a message (MESSAGE_STORE_ COMPLETE).
- It fails to store a message (MESSAGE_STORE_ FAILED).
- It successfully loads a message (MESSAGE_LOAD_ COMPLETE).
- It fails to load a message (MESSAGE_LOAD_ FAILED).

The name in brackets is the type of emitted event. For every cause of an event, a different type is used.
Furthermore, the component emits additional events and telemetry upon the success or failure of some internal operations. For a full definition, refer to 'Components/MessageStorage/MessageStorage.fpp'.

Name	Severity	Arguments	Description
MESSAGE_STORE_COMPLETE	activity low	storage_index: U32	A BBS message has been successfully written to the file system.
MESSAGE_STORE_FAILED	warning high	storage_index: U32 stage: MessageWriteError error_code: I32	An unhandled error while attempting to write a BBS message to the file system.
MESSAGE_LOAD_COMPLETE	activity low	storage_index: U32	A BBS message has been successfully read from the file system.
MESSAGE_LOAD_FAILED	warning high	storage_index: U32 stage: MessageReadError error_code: I32	An unhandled error while attempting to read a BBS message from the file system.
MESSAGE_INDEX_RESTORE_COMPLETE	activity low	storage_index: U32 index: U32	The index was restored from the storage directory by finding the highest index in use.
MESSAGE_INDEX_RESTORE_FAILED	warning high	storage_index: U32 stage: IndexRestoreError error_code: I32	An unhandled error while attempting to restore the index from the storage directory.

Name	Severity	Arguments	Description
INDEX_WRAP_AROUND	warning low		The index of BBS messages reached the maximum value of U32 and was thus reset to 0.
STORAGE_DIRECTORY_WARNING	warning low	directory: string size 128 was_able_to_create: bool	The component was not able to open the specified storage directory. E.g., because it does not exist.

4.5 Telemetry

Name	Data Type	Update	Description
NEXT_STORAGE_INDEX	U32	on change	The index at which the next message will be stored.
STORE_COUNT	U32	on change	The number of messages that have been attempted to be stored since the component was started.
LOAD_COUNT	U32	on change	The number of messages that have been attempted to be loaded since the component was started.

5 Implementation

5.1 Internal Design

5.1.1 Storage Format

Challenge

The component should store each message in a separate file. The component can assume that it is given access to a storage directory on the file system to store these files.
The component should not crash if the following events were to occur:

- Files other than the ones written by the components are (accidentally) put into the storage directory.
- Files the component wrote to the storage directory are corrupted. This is a plausible scenario as radiation can cause bit errors on the storage device.

Resulting Design Decision

Every message is stored with a unique file name in the storage directory (see *Indexing*).

Each message file follows the following format consisting of the following.

- Delimiter: A unique byte value that is expected as the first byte of every stored message file. Thus, we provide basic protection against trying to load files that do not originate from the MessageStorage component as message files.
- Message Length: A U32 in little-endian order that indicates how long the byte-serial representation of the message is. It helps to verify that the correct number of bytes is read and deserialized when loading the actual message from the file.
- Mesage Content: The byte-serial representation of the message data. It contains everything needed to fully restore a message so that the message object obtained from loading is the same as the one provided for storing n number of messages in reverse order of arrival to provide the capability to load the last n-stored messages.
- Assume that no index can be inferred from the message content.

Resulting Design Decision

The component assigns every message an index when the component starts an attempt to store the message. The provided index is a U32 and simply counted upwards every time a new message is about to be stored.

The index bijectively defines the file name. The file name format is "(id).bbsmsg," where '(id)' is replaced with the index number.

The index starts counting at 1 past the last index of a message found in the storage directory upon initialization of the component. If the component does not find any messages upon initialization, the index starts at 1. Restoring the index is implemented in *restoreIndexFromHighestStoredIndexFoundInDirectory()*.

For the sake of simplicity, we assume that the index counter never exceed the maximum value of U32. If it does, it is wrapped around to 0. This assumption is realistic as a defined U32 object can count over 4 trillion messages which are multiple magnitudes more than what we expect as defined in the mission success criteria.

5.1.2 Message Type Definition

Challenge

At the time this component was developed, the decision of what content will make up a message was still pending. Furthermore, this decision might change in the future.

Resulting Design Decision

The component encapsulates a message in the type BBSMessage. To make the component as independent of this type as possible, the component uses knowledge about this type only in a few very consciously chosen locations:

- The component received BBSMessages via its ports for communication with other components. This, the handler methods for port invocations know that the concrete type of a message is BBSMessage.
- In all other places of the component, the concrete type BBSMessage is hidden behind the abstract class Serializable. Consequently, in all of these places, the only assumption the component makes about the mesassages it is supposed to store is that they can be serialized into and deserialized from a raw byte buffer. No other assumptions are needed to store it in a file.

Therefore, the handler methods receive a message as a BBSMessage but only call other methods in the component by passing them a base class pointer of type Serializable to the message.

5.1.3 Serialization and Writing to File

Challenge

To write a message to a file, the message type must be serialized into a raw byte buffer. The address of that raw buffer must then be used to make a system call to the operating system.

Resulting Design Decision

- Serialize messages and other data by calling the framework's Serializable interface on the type which is to be serialized.
- Serialize data to a buffer that is allocated on the stack to avoid dynamic memory allocation. The buffer is implemented as a local class StackBuffer inside the MessageStorage component.