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

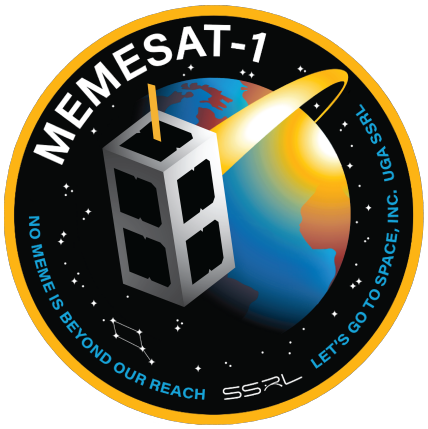
TlmChanWrapper Component Documentation

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
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
Initial Draft	Matthew Santoro	1.0.0
[04-01-2024] Updated for Feasibility Review. Added section for TlmChanWrapper.	Olivia Beattie	1.1.0
[04-04-2025] Preparing for IMR-3	Samuel Lemus	1.1.1

3 Purpose

This document aims to outline how F-Prime's telemetry database works, and how it will be utilized for MEMESat-1.

4 Overview

This component implements telemetry storage as a table accessed by the telemetry ID. That is, the Svc::TlmChan Component is used to store telemetry values written by other components. The values are stored in serialized form. The data is stored as a set of telemetry channels in a table. The data can be individually read back or periodically pushed to another component for transporting out of the system. Svc::TlmChan is an implementation of the Svc::TlmStore component in the Svc/Tlm directory.

The TlmChan component has two alternate implementations. One does a linear lookup to find the telemetry entry in the table based on the telemetry ID. This is more space efficient, but has slower performance because the table is traversed each time. The second uses the telemetry ID as an index into the table. It is faster but can be space inefficient if there are disjoint telemetry IDs. Which version is used can be selected by putting the desired file in the

mod.mk file.

Description of directory files:

- TelemChannelImpl.hpp(.cpp) - implementation of the common functions of the telemetry storage.
- TelemChannelImpIndex.cpp - implements lookup by using the telemetry ID as an array index.
- TelemChanLookup.cpp - implements lookup by traversing the table and looing for the ID.
- TelemChanTask.cpp - implements the rate group handler to write the telemetry to the downlink.
- TlmChanImplCfg.hpp - Contains configuration values for the component.

5 TlmChan Design

5.1 Ports

Port Data Type	Name	Direction	Kind	Usage
Svc::Sched	Run	Input	Asynchronous	Execute a cycle to write changed telemetry channels.
Fw::Tlm	TlmRecv	Input	Synchronous Input	Update a telemetry channel.
Fw::Tlm	TlmGet	Input	Synchronous Input	Read a telemetry channel.
Fw::Comm	PktSend	Output	n/a	Write a set of packets with updated telemetry.

The Svc::TlmChan component has an input port TlmRecv that receives channel updates from other components in the system. These calls from the other components are made by the component implementation classes, but the generated code in the base classes takes the type specific channel value and serializes it, then makes the call to the output port. The component can then store the channel value as generic data. The channel values are stored in an internal double-buffered table, and a flag is set when a new value is written to the channel entry.
The table is a structure that holds a custom TlmEntry structure.

```
typedef struct tlmEntry {
    FwChanIdType id;            //!< telemetry id stored in slot
    bool updated;               //!< set whenever a value has been written.
                                // Used to skip if writing out values for
                                // downlinking.
    Fw::Time lastUpdate;        //!< last updated time
```

```
    Fw::TlmBuffer buffer;      //!< buffer to store serialized telemetry
    tlmEntry* next;            //!< pointer to next bucket in table
    bool used;                 //!< if entry has been used
    NATIVE_UINT_TYPE bucketNo; //!< for testing
} TlmEntry;
struct TlmSet {
    TlmEntry* slots[TLMCHAN_NUM_TLM_HASH_SLOTS]; //!< set of hash slots
                                                  // in hash table
    TlmEntry buckets[TLMCHAN_HASH_BUCKETS];     //!< next free bucket
    NATIVE_INT_TYPE free;
} m_tlmEntries[2];
```

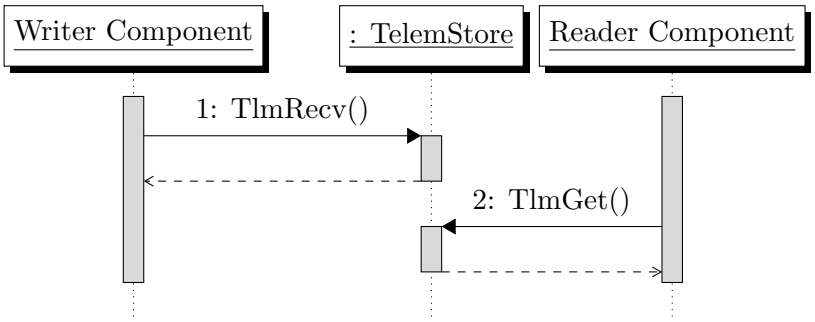
It looks like there is a maximum number of telemetry message types that we can store; the default max is 50. The TlmChanImplCfg.hpp file at config/TlmChanImplCfg.hpp has the definitions for the sizes of the components and the number of buckets. We will need to modify this based on the number of components that we have and the amount of telemetry that we want to store. The Run_handler will work with the scheduler and not the rate group to execute the functions of this component.

When a request is made for a nonexistent channel, the call will return with an empty buffer in the Fw::TlmBuffer value argument. This is to cover the case where a channel is defined in the system, but has not been written yet. If the channel has not ever been defined, there is no way to programmatically determine that from the TlmGet port call.

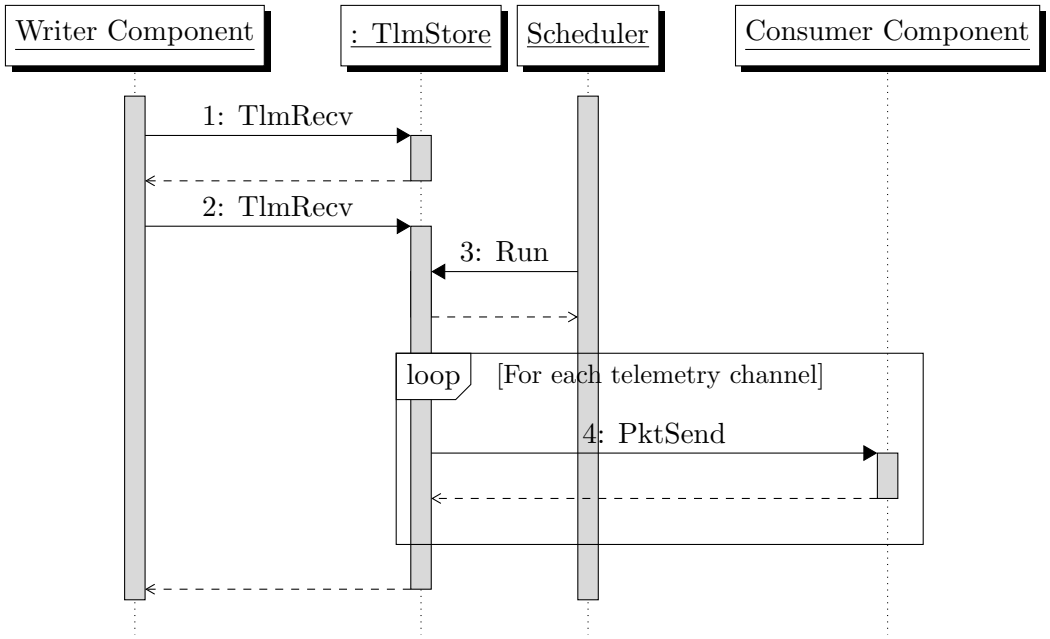
The implementation uses a hashing function that is tuned in the configuration file TlmChanImplCfg.hpp.

6 Diagrams

6.1 Database Scenario



6.2 External User Scenario



7 TlmPacketizer Overview

The Svc::TlmPacketizer Component is used to store telemetry values written by other components. The values are stored in serialized form. TlmPacketizer differs from Svc::TlmChan in that it stores telemetry in defined packets instead of streaming the updates as they come. The defined packets are passed in as a table to the setPacketList() public method. When telemetry updates are passed ot the component, they are placed at the offset in a packet buffer defined by the table. When run() port is called, all the defined packets are sent to the output port with the most recent telemetry values. This is meant to replace Svc::TlmChan for use cases where a more compact packet form is desired. The disadvantage is that all channels are pushed whether or not they have been updated.

8 TlmPacketizer Design

8.1 Ports

Port Data Type	Name	Direction	Kind	Usage
Svd::Sched	Run	Input	Asynchronous	Execute a cycle to write changed telemetry channels.
Fw::Tlm	TlmRecv	Input	Synchronous Input	Update a telemetry channel.
Fw::Com	PktSend	Output	n/a	Write a set of packets with updated telemetry.

The component has an input port TlmRecv that receives channel updates from other components in the system. These calls from the other components are made by the component implementation classes, but the generated code in the base classes takes the type specified channel value and serializes it, then makes the call to the output port. The component can then store the channel value as generic data. the channel ID is used to look up offsets for the channel in each of the defined packets. A channel can be defined in more than one packet. The time tag is stripped from the incoming telemetry value. The time tag of the channel will become the time tag of the entire frame when it is sent.

The implementation uses a hashing function to find the location of telemetry channels that is tuned in the configuration file TlmPacketizerImplCfg.hpp.
When a call to the Run() interface is called, the packet writes are locked and all the packets are coped to a second set of packets. Once the copy is complete, the packet writes are unlocked. The destination packet set gets updated with the current time tag and is sent out in the pktSend() port.

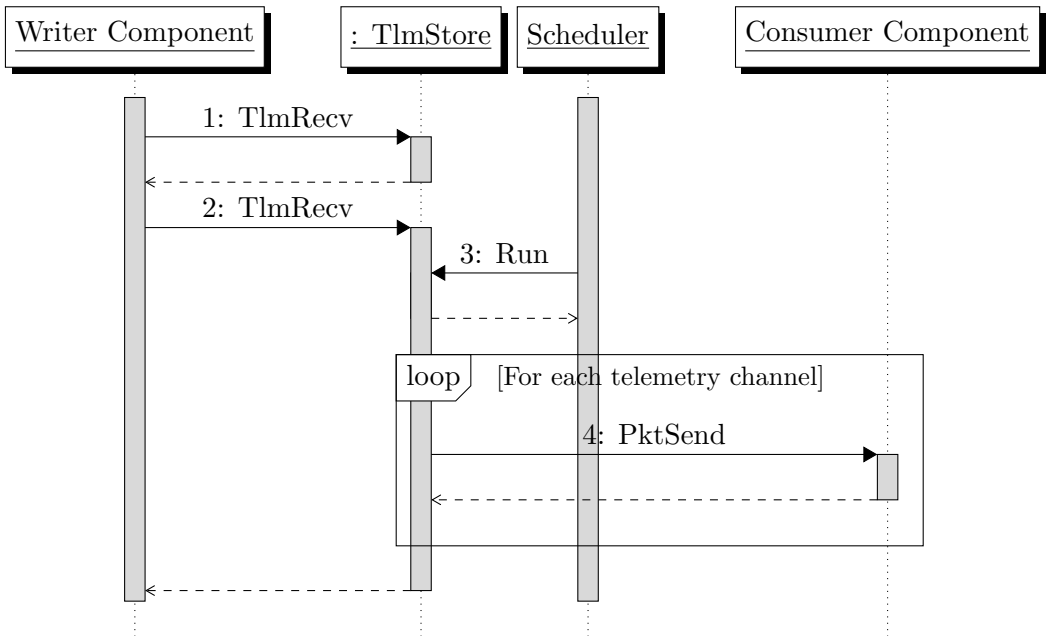
A configuration value in the TlmPacketizerImplCfg.h defines a set of hash buckets to store the telemetry values. The number of buckets has to be at least as large as the number of telemetry channels defined in the system. The number of channels in the system can be determined by invoking make comp_report_gen from the deployment directory. The number of table slots and the hash value in the configuration file can be varied to balance the amount of memory for slots versus the distribution of buckets to slots. See TlmPacketizerImplCfg.h for a procedure on how to tune the algorithm.

Note: there are three packet update modes:

- PACKET_UPDATE_ALWAYS - always send packets, even if no changes to channel data.
- PACKET_UPDATE_ON_CHANGE - only send packets if any of the channels updated.
- PACKET_UPDATE_AFTER_FIRST_CHANGE - always send packets, but only after the first channel has been updated.

8.2 Diagrams

8.2.1 TlmPacketizer External User Option (shows the scenario where telemetry channels are written to packets.)



9 TlmChanWrapper Overview

The telemetry database is interacted with through a telemtry channel wrapper component (TlmChanWrapper) that is connected to the Scheduler. The wrapper acts as a medium between the TlmChan component pre-defined by F Prime and the Scheduler component in order to preserve the integrity of the predefined code while also created a small layer of abstraction with regards to our custom scheduler.

10 TlmChanWrapper Design

10.1 Ports

Port Data Type	Name	Direction	Kind	Usage
Svc::Sched	tlmChanOut	Output	N/A	Port to the TlmChan to dowlink the stored telemetry values.
Scheduler SendSchedulerPort	sendSchedule	Output	N/A	Port to send a schedule to the Scheduler Component.
Scheduler RunSchedulerPort	scheduledHandler	Input	Asynchronous	Port from the Scheduler to run the component on the defined timeline.