***Author**: Nyameaama Gambrah* **Team:** *Vehicle Firmware*

PTAM

- storeString():
- storeDouble():
- storeInt():
- getStringData():
- getDoubleData():
- getIntData():
- clearData():
- clearAllData():
- getLastString():
- getLastDouble():
- getLastInt():

Appropriate use of PTAM

PTAM API functions can be designated into different categories; data store functions, data retrieval functions and utility functions

The data store functions include:

- storeString
- storeDouble
- storeInt

The data retrieval functions include:

- getStringData
- getDoubleData
- getIntData

This document will outline the appropriate use of PTAM in all flight firmware modules.

Instantiating PTAM

PTAM can be used anywhere in the firmware stack where the PTAM module is adequately called and defined. See below an example of a PTAM instantiation:

```
#include “_ptam.h”

SharedMemory& sharedMemory = SharedMemory::getInstance();
```

***Author**: Nyameaama Gambrah* ***Team:** Vehicle Firmware*

Using data store functions

The PTAM data store functions were developed to store multiple values in one PTAM register. This means that a PTAM register can be used as a "variable" or a collection of data values ("array"). This is the default behavior and has to be kept in mind.

EXAMPLE_PTAM_REGISTER (*variable implementation)*

DataPoint

EXAMPLE_PTAM_REGISTER_2 (*collection of data implementation)*

DataPoint1	DataPoint2	DataPoint3	DataPoint4

The example above showcases how PTAM registers can be used as a variable implementation or a collection of data implementation (array).

Keeping this in mind, this implementation also means that when data is added to a PTAM register, it is automatically appended to any data already in the register. In specific use cases such as a collection of data (array) implementation, this behavior is desirable. However if a PTAM register intended use case is a variable implementation, you **<u>must</u>** clear data from the PTAM register on system call. See example below:

```
SharedMemory& sharedMemory = SharedMemory::getInstance();

//Clear previous register to avoid memory overflow

sharedMemory.clearData("stateDescript");

//Put new data in

sharedMemory.storeString(std::string("stateDescript"),std::string("ARMED"));
```

In this example we instantiate an instance of PTAM with the intent of changing the PTAM register "stateDescript" with an updated value. We have to clear the previous value in the register because if we do not, the register will be continuously appended every program cycle which can lead to memory overflow issues.

Using data retrieval functions

PTAM also includes data retrieval functions. These functions include:

- getStringData():

- getDoubleData():
- getIntData():

The retrieval functions return a vector of appropriate type. This is done by default as PTAM by design, stores a collection of data values. Shown below is an example in which a register is referenced and the values are output.

```
std::vector<std::string> stringData = obj -> getStringData("stateDescript");

for (const std::string& element : stringData) {

        std::cout << element << std::endl;

}
```

PTAM API has additional functions to make it easier to retrieve data from variable implementations of PTAM register. These functions include:

- getLastDouble()
- getLastInt()
- getLastString()

The getLast functions make it easier to interface with registers by returning the last value appended. This means that the register does not have to be traversed to get the most updated value.

PTAM is a powerful tool that can be used to improve the efficiency of writing flight firmware and it works by creating a shared memory space that is accessible to all of the modules. When a module needs to store data, it can write the data to the shared memory space. Other modules can then read the data from the shared memory space by referencing the unique ID of the data.