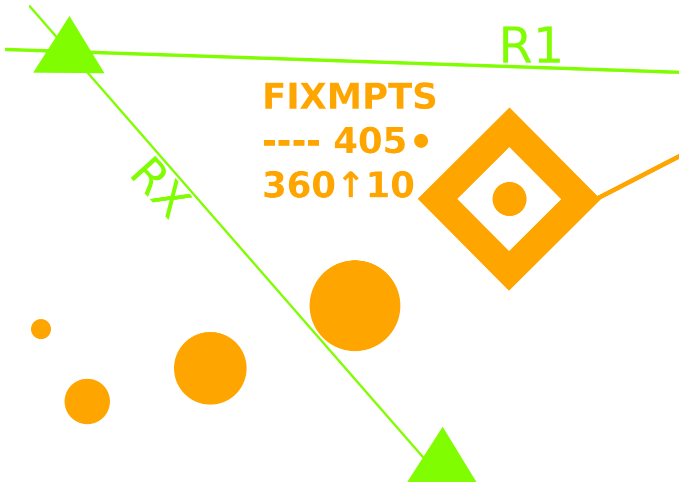

Document ID: FixMPTS-AsterixLib-TN-001

Version 0.1

August 2021

Revision History

Name	Date	Reason For Change	Version
F. Kreuter	01.08.2021	Initial Version	V 0.1

Contents

1 Purpose

This document is to provide an overview of the code and build structure. Both items will be covered in the first part. The second part will be focusing on a few examples which are meant to provide some idea on how the library could be extended or used in other projects.

The intended reader group for this document are developers who want to contribute to the project or those who plan to integrate the library into another product. In order to fully understand this document it is expected that the reader has some experience with C++ [1] and its standard library as well as with the build system Apache Ant [2]

The following chapters will focus on the code. How to get it, what you get, how to build it and where to find the relevant documentation. So lets start with how to get the code.

2 References

List of documents this technical note document has references to.

[1] Stroustrup, Bjarne *The C++ Programming Language (Fourth ed.)*.
2013, Addison-Wesley. ISBN 978-0-321-56384-2

[2] Apache Software Foundation *Apache Ant*
[https://ant.apache.org/]
Version 1.10.11, Date 2021-07-13

[3] Git Community *Git SCM*
[https://git-scm.com/]
Version 2.32.0, Date 2021-06-06

3 Set Up

The entire project code is version controlled with git [3].At the moment the prime repository is hosted in github [?]. To clone the repository no account is needed. Only for contribution one would need an account with github. In order to get the code either download it from Github directly https://github.com/FixMPTS/FixMPTSAsterixLib, or use the means provided by git. To do so, on the terminal of your choice navigate to the directory where you want the code to reside and run the following command

```
git clone git@github.com:FixMPTS/FixMPTSAsterixLib.git
```

This should give you some new directories and files which top level structure will look like 1.

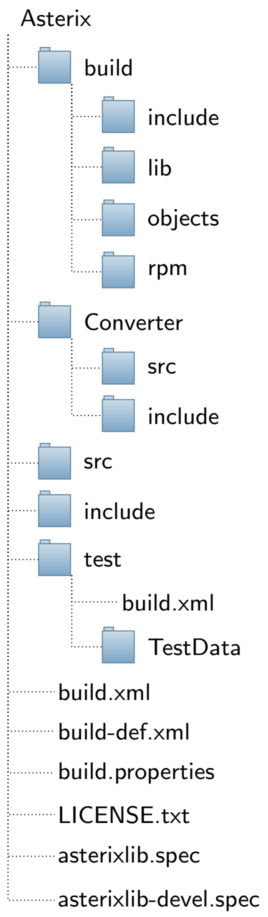

Figure 1: Repository folder structure

We will explain the content and purpose of each of the important files and directories in a minute. A detailed reading how to contribute and which other git commands are allowed please also read chapter 6.

build: In the checkout this directory will only contain the listed sub directories. All are empty. Those directories are only meant to distribute the folder structure required during the build process. All files within this directory and its sub directories are ignored in git.

Converter: This directory holds the static converter functions. It is split into two sub directories. The include directory contains all header files. Complementary the second sub directory, src, holds all source files containing the implementation.

src: This directory holds all source files for the core library. Currently no sub directories are foreseen for the library.

include: Purpose of this directory is to hold all core header files. All header files within this directory will later be packed into the package for this library.

test: This folder contains the source and header files for the unit tests. In addition it comes with a sub folder named **TestData** which holds all the input data for the unit tests.

Alongside those directories there are a few files located top level. Those are all required for building. Files starting with build belong to the ANT build system. The usage will be described in chapter 7. On the other hand, files ending with spec define how the rpm package of the library should be constructed. A detailed read on packaging can be found in chapter 4

4 Build System

Compiling and packaging of the library is done with Apache Ant [2]. All the configuration needed to trigger the build cam be found in the *build.xml* file and the *build.properties* file. As the names already suggest, the *build.properties* file holds the variables and other properties needed for the build, whereas the *build.xml* file holds the instructions on how to call the compiler and linker. In order to build, one has to go to the top level directory of the checkout. From there, on the terminal, call

```
ant all
```

given Apache Ant is installed with the *ant-contrib* and *cpptask* modules. The build target **all** will compile the source code, link everything together, and afterwards package the library and include files to dedicated RPM packages. Besides the **all** target the following set of build targets is defined at the moment:

- all: compile, link and RPM packaging
- clean: remove all build artefacts
- libs: will just compile and link the library. No packaging of the build artefacts
- rpm: packaging of the artefacts into RPM packages
- rpm.devel: packaging of the devel package containing the include files for the library
- test: build and run the unit tests
- asterix.compile: compilation of the source files without linking the result
- asterix.build: compile the source files and link everything into the library file

For more details on how, where and when the version of the different build artefacts, please refer to chapter 5.

5 How and when to change version numbers

In general not every tiny change needs an update to the version number. That said, every time the library is released, the version number has to be increased. The rule which part of the number changes when is as follows. If the new release just contains minor bug fixes without changes to the interfaces, only the minor part of the version number needs to be increased. For those changes which affect the interfaces or implement new features an increase of the major part of the version number should be applied.

To change the number one has to do this in the top level *build.xml* file. There the number has to be changed for the **asterix.buildlib** target as well as for the two RPM generations targets, **rpm** and **rpm.devel**.

6 Git and Branching

The very first thing is to clone the repository. On the command line navigate to the folder where the source code of the library should end up. There use the following command

```
git clone git@github.com:FixMPTS/FixMPTSAsterixLib.git
```

to clone the repository. In order to update the source code toe the latest state, on the command line call

```
git checkout <branch>
git pull --rebase
```

Replace *¡branch¿* with whatever release or master branch the code should be of.

Providing code is as easy as checking it out. First of all it is important the source branch is up to date. If it is not pulling the latest state is recommended. After that create a new branch for the changes by running the *checkout* command

```
git checkout -b <issue no.>_<additional_name>
```

Where *¡issue no.¿* represent the number of the issue fixed by the code changes or the number of the issue handling the newly implemented feature implemented by the change. And *¡additional_name¿* is some additional information for a more easy recognition on what the change is about.Now commit all changes and afterwards push them to the remote repository with

```
git push
```

It is recommended to use **git gui** for committing, as it provides line wise commits allowing fore a more atomic and hopefully self contained simple commits. As soon as you are happy with the code the changes you made you are in the position to file a *pull request* from your branch to the remote branch the local branch is based on. Filing the *pull request* will trigger a review of the changes and if all is according to expectationa nd sound the change be included in the next release.

7 Using the Library

Integrating the library into an existing project is pretty simple. The only thing required is installing the two RPM packages (library and devel). The installation already chooses directories which can be discovered by the system without the need to set any variables. Just as information the current target directories are */usr/lib* and */usr/include/asterix*. To make use of the library use the following compile and linger arguments for the GCC and Clang compiler suits.

```
compile: -I /usr/include/asterix
link: -lasterix
```

For the final product just add the package dependency.

8 Examples

This section contains a few examples on how to use the library. The code of all examples will be discussed in the corresponding chapter. In the following sections we provide examples covering encoding, decoding, and implementing a new category.

8.1 Encoding

As o now encoding is not implemented for all Asterix Categories. Therefore we will demonstrate how to encoding data with the help of Asterix category 62.

```
 1  #include "AsterixCategory062.h"
 2  #include "TrackTypeIf.h"
 3  #include "TrackDetectionAges.h"
 4
 5  #include <map>
 6  #include <vector>
 7  #include <string>
 8  #include <memory>
 9
10  public void encodeCAT62(){
11      AsterixCategory062 message = AsterixCategory062();
12
13      // Detection ages only relevant for track messages
14      std::shared_ptr<AsterixItemMaxAges> max_ages = std::make_shared<AsterixItemMaxAges>();
15      max_ages->setMaxAgePerType( DetectionEntry::DET_TYPE::PSR, 10.0 );
16      max_ages->setMaxAgePerType( DetectionEntry::DET_TYPE::SSR, 10.0 );
17
18      // set some items
19      std::map<std::string, unsigned char> non_track_values;
20      non_track_values["I062/010:SAC"] = (unsigned char) 1;
21      non_track_values["I062/010:SIC"] = (unsigned char) 123;
22      message.setNonTrackRelatedvalues( non_track_values );
23
24      // Add the track information too
25      std::shared_ptr<TrackTypeIf> track = std::make_shared<TrackTypeIf>( 10 );
26
27      // Set the items to be contained in the output. Mandatory items are anyway encoded
28      std::map<std::string, bool> items;
29      items["I062/010"] = true; // SDPS ID
30      items["I062/040"] = true; // Track number
31
32      // Encode the message
33      std::vector<unsigned char> msg = message.getEncodedMessage( track, items, max_ages );
34  }
```

Listing 1: Encoding example

The encoding function comes with a few inputs not necessarily clear at the beginning, which are highly depend on the type of Asterix message to be encoded. So lets go right into the code. Lets start with

```
11  AsterixCategory062 message = AsterixCategory062();
```

Initialising the new Asterix message. From this point forward we can start filling in all the relevant information.

```
14  std::shared_ptr<AsterixItemMaxAges> max_ages = std::make_shared<AsterixItemMaxAges>();
15  max_ages->setMaxAgePerType( DetectionEntry::DET_TYPE::PSR, 10.0 );
16  max_ages->setMaxAgePerType( DetectionEntry::DET_TYPE::SSR, 10.0 );
```

In order to decide whether a detection information is still up to date or is outdated, the Asterix encoder needs to know what the expected maximum age per detection type is. This has also on influence of the detection state, special in CAT 62. For this the maximum ages are defined within the *AsterixItemMaxAges* type. Usually this type is derived from configuration and thus there is no need to set it up again for every single message.

```
        std::map<std::string, unsigned char> non_track_values;
        non_track_values["I062/010:SAC"] = (unsigned char) 1;
        non_track_values["I062/010:SIC"] = (unsigned char) 123;
        message.setNonTrackRelatedvalues( non_track_values );
```

As Asterix category is meant to encode track data into Asterix, most of the items of this category can be filled from the attributes found in the *TrackTypeIf* type. Those four lines above deal with those values not directly retrievable from the track. Providing the items is straight forward as map of key, value pairs. The key is current a string although it will be replaced by a constant in the future. The values are the bytes to be added. Maybe the encoded you want to use comes with a more convenient interface so watch out for the documentation. After all items have been defined the message can be updated.

```
        std::shared_ptr<TrackTypeIf> track = std::make_shared<TrackTypeIf>( 10 );
```

In above text it has been mentioned already, the main type from which most of the items of the category are retrieved is the *TrackTypeIf* type. This interface is meant to be exchanged between the Asterix library and the SPDS using it.

```
        std::map<std::string, bool> items;
        items["I062/010"] = true; // SDPS ID
        items["I062/040"] = true; // Track number
```

Besides the mandatory items each Asterix message requires, there are a set of optional items. From the data providing input types it is not possible to determine which items shall be contained in the message. In addition one might want to have the same set of items in each messages, probably defined in the Asterix Service configuration, the set of items expected can be provided to the encoding function. This definition is a simple mapping of item name and flag whether it is expected or not. For simplicity only items that shall be present need to be defined.

```
        std::vector<unsigned char> msg = message.getEncodedMessage( track, items, max_ages );
```

Finally, everything required for encoding the message is set up and we can go ahead with encoding the message. The encode function returns the vector containing the bytes of the message. This vector can be pushed to network directly without the need of further processing.

The example in the previous paragraph is one of the most simple examples containing all required information to get a basic Asterix SDPS message. Although it might look complicated to generate all the different variables steering the encoding, most of this work has to be done only once. For example the *items*, *max_ages*, and most of the*non_track_values* have to be created only once, as they most likely stay constant and will be re-used for every single message.

8.2 Decoding

The second half of the tasks to be done bu the Asterix Library, is decoding of incoming messages from the various sensors or for displaying measurements. On of the most simple examples, still containing all the relevant information, will be presented in listing below.

```
        #include "AsterixCategory020.h"
        #include "BinaryHelper.h"

        public void decodeCat020(){
           unsigned int block_length = readLength(as_socket); // readLength needs to be implemented
           char astx_block[block_length];
           read( as_socket, astx_block, block_length ); // Needs to be implemented too
           int remaining_length = sizeof(astx_block);
           std::deque<char> message( astx_block, astx_block + sizeof(astx_block) / sizeof(char) );
           AsterixCategory020* cat020_message = new AsterixCategory020( remaining_length, message );

           cat020_message->printMessage();

           cat020_message->getFspecString(); // "e.g. 1111111111111111001111111100"
           cat020_message->getValue( "I020/010#SAC" ); //e.g. "98"
           cat020_message->getValue( "I020/010#SIC" ); // e.g. "102"
        }
```

Listing 2: Decoding example

In the above example it is expected the incoming data arrive via a network socket. How to read the data from the socket is up to the developer and not part of this example. The same goes for making sure the Asterix Category of the received data matches the one we try to decode as well as reading the length of the Asterix message from the data and performing all the sanity checks needed. Now with the prerequisites out of the way lets dive straight into the exmaple.

```
        char astx_block[block_length];
```

This will just create the input buffer to store the bytes read from the network before decoding them.

```
        std::deque<char> message( astx_block, astx_block + sizeof(astx_block) / sizeof(char) );
        AsterixCategory020* cat020_message = new AsterixCategory020( remaining_length, message );
```

Above two lines already do all the decoding. The first line copies the byte into a queue, the input format required for the decoding. Constructing the new Asterix message by providing the binary input data along with the number of bytes to be decoded, will already trigger the decoding. After this point we are able to access the decoded items and sub items of the message.

```
        cat020_message->getFspecString(); // "e.g. 11111111111111100111111100"
        cat020_message->getValue( "I020/010#SAC" ); //e.g. "98"
        cat020_message->getValue( "I020/010#SIC" ); // e.g. "102"
```

Accessing the various items of the Asterix message is fairly straight forward. One only needs to call the *getValue()* method with the name of the item being the only parameter. This will return the string representation of the corresponding value or through an std::out_of_range exception if there is no item contained within the message.

8.3 Implementing a new Asterix Category

In this example we will explain the steps one has to take to implement an entirely new Asterix Category.

To start a new Asterix Category, in this example we will call it CAT 000, all we have to do us to create a new class called *AsterixCat000* and inherit from *AsterixCategory*. There are a few methods which need to overwritten as they are purely abstract with *AsterixCategory*. These methods are:

- void setUAP()
- void setSubitems()

Both methods are the once that actually define the Asterix Category. So lets put the few basic things just described together.

```
        #include "AsterixCategory.h"

        class AsterixCategory000 : public AsterixCategory{
            private:
                void setUAP() override;
                void setSubitems override;

            public:
                AsterixCategory000();
        }
```

Listing 3: AsterixCategory000.h

The implementation will look like follows.

```
        #include "AsterixCategory000.h"
        #include "AsterixCommonDef.h"
        #include "AsterixItemFixedLength.h"
        #include "AsterixItemVariableLength.h"
        #include "AsterixSubitemUnsigned.h"
        #include "AsterixSubitemBitNamed.h"

        AsterixCategory000::AsterixCategory000(){
            // nothing to do at the moment
        }

        void AsterixCategory000::setUAP(){
            uap.clear();
            uap.insert( UAP_Item_T( 1, std::make_shared<AsterixItemFixedLength>( "I000/010", 2 ) ) );
            uap.insert( UAP_Item_T( 2, std::make_shared<AsterixItemVariableLength>( "I000/020" ) ) );
        }

        void AsterixCategory000::setSubitems(){
           //Sensor Identification
           subitem_map_t sensor_identification;
           sensor_identification.push_back(
              subitem_t( "I000/010#SAC",
                 std::make_shared<AsterixSubitemUnsigned>( 8, CommonConverter::NoneConverter ) ) );
           sensor_identification.push_back(
              subitem_t( "I000/010#SIC",
                 std::make_shared<AsterixSubitemUnsigned>( 8, CommonConverter::NoneConverter ) ) );

           // Target Report descriptor
           subitem_map_t target_report_descriptor;
```

```
            target_report_descriptor.push_back(
               subitem_t( "I000/020#TYP",
                  std::make_shared<AsterixSubitemBitNamed>( 1, CommonConverter::NoneConverter,
                     AsterixSubitemBitNamed::value_names_t( { { 0, "T1" }, { 1, "T2" } } ) )));
            target_report_descriptor.push_back(
               subitem_t( "I000/020#FLG",
                  std::make_shared<AsterixSubitemBitNamed>( 1, CommonConverter::NoneConverter,
                     AsterixSubitemBitNamed::value_names_t( { { 0, "FLG1" },
                        { 1, "FLG2" } } ) ) ) );

            //Add all the sub items to the category sub item map
            subitems.insert( subitem_map_item_t( 1, sensor_identification ) );
            subitems.insert( subitem_map_item_t( 2, target_report_descriptor ) );
         }
```

Listing 4: AsterixCategory000.cpp

In the above example there are a few lines we did not explain before so lets have a closer look at them.

```
         uap.clear();
         uap.insert( UAP_Item_T( 1, std::make_shared<AsterixItemFixedLength>( "I000/010", 2 ) ) );
```

The protected variable *uap* inherited from the parent class is defined as map of *unsigned int* and *AsterixItem**, where the key is supposed to be the FRN of the Asterix Item. For convenience reasons the map is defined as dedicated data type called *UAP_Item_T*. For item *I000/010* the Asterix Items is defined with its name and its length as second parameter to the constructor of the item. Now, that the high level structure with all the Asterix Items is defined, lets move on to the more detailed description of the new Asterix Category by laying out the definition of each individual item.

```
         subitem_map_t sensor_identification;
         sensor_identification.push_back(
            subitem_t( "I000/010#SAC",
               std::make_shared<AsterixSubitemUnsigned>( 8, CommonConverter::NoneConverter ) ) );
```

Per item we define a variable of type *subitem_map_t*. This is just a short cut for a vector consisting of a pair of string (item name) and *AsterixSubItem*. In the example above this is an unsigned integer of length 8 bit with no special converter. The converter parameter to the AsterixSubItem constructor defines the translation from the raw bits to the expected output format and vice versa.

```
         subitems.insert( subitem_map_item_t( 1, sensor_identification ) );
```

After all sub items are created it is time to add them to the inherited sub items list *subitems*. From this point onwards our basic AsterixCategory is defined.

But, so far the AsterixCategory is not very useful, as it currently is not able to decode or encode any Asterix messages. So lets define the decoding part first. When looking at the *AsterixCategory* class, one thing to spot is the missing encode(...) function. This is because decoding with the Asterix Library is done in the constructor of the dedicated Asterix Category. For that we add the following code to our class.

```
         AsterixCategory000(int length, std::deque<char>& m_queue);
```

Listing 5: AsterixCategory000.h

Followed by its implementation.

```
         AsterixCategory000::AsterixCategory000(int length, std::deque<char>& m_queue):
             AsterixCategory( 1, length, m_queue ){
             initCategory();
         }
```

Listing 6: AsterixCategory000.cpp

From Listing: 6 it is clearly visible how simple the decoding is. Calling *initCategory()*, which is implemented in the parent class, does four things.

1. calls setUAP()
2. calls setSubitems()
3. calls readFspec()
4. calls decode()

After this point we are able to access any item in its decoded form.

Encoding of the message on the other hand is a bit more work. As this depends highly on the input type, nothing is done for us in the parent class. So lets define the input type first by adding the following code to the beginning of the two files we already created.

```
        class EncodedType{
            unsigned int sac;
            unsigned int sic;
            unsigned short t1;
            unsigned short t2;

            EncodedType();
            EncodedType( unsigned int sa, unsigned int si, unsigned short type1,
                unsigned short type2);
            unsigned int getSac();
            unsigned int getSic();
            unsigned short getT1();
            unsigned short getT2();
        }
```

Listing 7: AsterixCategory000.h

With the following implementation.

```
        EncodedType::EncodedType(){
            sac = 0;
            sic = 0;
            t1 = 0;
            t2 = 0;
        }

        EncodedType::EncodedType(unsigned int sa, unsigned int si,
            unsigned short type1, unsigned short type2):
            sac(sa), sic(si), t1(type1), t2(type2){
        }
        unsigned int EncodedType::getSac(){ return sac; }
        unsigned int EncodedType::getSic(){ return sic; }
        unsigned short EncodedType::getT1(){ return t1; }
        unsigned short EncodedType::getT2(){ return t2; }
```

Listing 8: AsterixCategory000.cpp

With the implementation of the input type we can focus on encoding the Asterix Message by adding the *encode()* function listed below.

```
        std::vector<unsigned char> getEncodedMessage( EncodedType input );
```

Listing 9: AsterixCategory000.h

Which comes with the following implementation.

```
        std::vector<unsigned char> AsterixCategory000::getEncodedMessage( EncodedType input ){
            std::vector<unsigned char> header;
            std::vector<unsigned char> message;

            for( auto item : fpsec_item_name_map ) {
              // Reset the FSPEC for this item
              fspec[item.first] = false;

              if( item.second == "I000/010" ) {
                 fspec[item.first] = true;
                 message.push_back(input.getSac());
                 message.push_back(input.getSic());
              }

              if( item.second == "I000/020" ) {
                 fspec[item.first] = true;
                 std::bitset<8> item( 0 );
                 item[0] = input.getT1() > 0 ? 1 : 0;
                 item[1] = input.getT2() > 0 ? 1 : 0;
                 message.push_back(item.to_ulong() & 0xff);
              }
            }

            // Add the header
            getHeader( header, message.size() );
            message.insert( message.begin(), header.begin(), header.end() );
            return message;
        }
```

Listing 10: AsterixCategory000.cpp

Although the above code looks like there is a lot to do for just encoding two items, in the end it is a fairly straight forward task. The most important thing to remember probably to generate the header after the rest of the message has been set up. This is due to the size of the message being a mandatory item in the header. With this information we can make use of the *getHeader(..)* function defined in the parent class. Everything else is just about setting the proper item in the *fspec* to true and encoding the bytes with the correct value.

For completeness the entire code of the example in one listing.

```
        #include "AsterixCategory.h"

        #include <vector>
        #include <bitset>
        #include <deque>

        /** Definition of the input type to encode the Asterix Message from */
        class EncodedType{
            unsigned int sac;
            unsigned int sic;
            unsigned short t1;
            unsigned short t2;

            EncodedType();
            EncodedType( unsigned int sa, unsigned int si, unsigned short type1,
                unsigned short type2);
            unsigned int getSac();
            unsigned int getSic();
            unsigned short getT1();
            unsigned short getT2();
        }

        class AsterixCategory000 : public AsterixCategory{
            private:
                void setUAP() override;
                void setSubitems override;
                std::vector<unsigned char> getEncodedMessage( EncodedType input );

            public:
                AsterixCategory000();
                AsterixCategory000(int length, std::deque<char>& m_queue);
        }
```

Listing 11: AsterixCategory000.h

Which will result in the following implementation

```
        #include "AsterixCategory000.h"
        #include "AsterixCommonDef.h"
        #include "AsterixItemFixedLength.h"
        #include "AsterixItemVariableLength.h"
        #include "AsterixSubitemUnsigned.h"
        #include "AsterixSubitemBitNamed.h"
        #include <memory>

        EncodedType::EncodedType(){
            sac = 0;
            sic = 0;
            t1 = 0;
            t2 = 0;
        }

        EncodedType::EncodedType(unsigned int sa, unsigned int si,
            unsigned short type1, unsigned short type2):
            sac(sa), sic(si), t1(type1), t2(type2){
        }
        unsigned int EncodedType::getSac(){ return sac; }
        unsigned int EncodedType::getSic(){ return sic; }
        unsigned short EncodedType::getT1(){ return t1; }
        unsigned short EncodedType::getT2(){ return t2; }

        ///////////////// AsterixCat000 related methods
        AsterixCategory000::AsterixCategory000(){
            // nothing to do at the moment
        }

        AsterixCategory000::AsterixCategory000(int length, std::deque<char>& m_queue):
            AsterixCategory( 1, length, m_queue ){
            initCategory();
        }

        void AsterixCategory000::setUAP(){
            uap.clear();
            uap.insert( UAP_Item_T( 1, std::make_shared<AsterixItemFixedLength>( "I000/010", 2 ) ) );
            uap.insert( UAP_Item_T( 2, std::make_shared<AsterixItemVariableLength>( "I000/020" ) ) );
        }

        void AsterixCategory000::setSubitems(){
           //Sensor Identification
           subitem_map_t sensor_identification;
           sensor_identification.push_back(
              subitem_t( "I000/010#SAC",
                 std::make_shared<AsterixSubitemUnsigned>( 8, CommonConverter::NoneConverter ) ) );
           sensor_identification.push_back(
              subitem_t( "I000/010#SIC",
                 std::make_shared<AsterixSubitemUnsigned>( 8, CommonConverter::NoneConverter ) ) );

           // Target Report descriptor
           subitem_map_t target_report_descriptor;
           target_report_descriptor.push_back(
              subitem_t( "I000/020#TYP",
                 std::make_shared<AsterixSubitemBitNamed>( 1, CommonConverter::NoneConverter,
                    AsterixSubitemBitNamed::value_names_t( { { 0, "T1" }, { 1, "T2" } } ) )));
           target_report_descriptor.push_back(
              subitem_t( "I000/020#FLG",
                 std::make_shared<AsterixSubitemBitNamed>( 1, CommonConverter::NoneConverter,
                    AsterixSubitemBitNamed::value_names_t( { { 0, "FLG1" },
                       { 1, "FLG2" } } ) ) ) );

           //Add all the sub items to the category sub item map
           subitems.insert( subitem_map_item_t( 1, sensor_identification ) );
           subitems.insert( subitem_map_item_t( 2, target_report_descriptor ) );
        }

        std::vector<unsigned char> AsterixCategory000::getEncodedMessage( EncodedType input ){
            std::vector<unsigned char> header;
            std::vector<unsigned char> message;

            for( auto item : fpsec_item_name_map ) {
              // Reset the FSPEC for this item
              fspec[item.first] = false;

              if( item.second == "I000/010" ) {
                 fspec[item.first] = true;
                 message.push_back(input.getSac());
                 message.push_back(input.getSic());
              }

              if( item.second == "I000/020" ) {
                 fspec[item.first] = true;
                 std::bitset<8> item( 0 );
                 item[0] = input.getT1() > 0 ? 1 : 0;
                 item[1] = input.getT2() > 0 ? 1 : 0;
                 message.push_back(item.to_ulong() & 0xff);
              }
            }

            // Add the header
            getHeader( header, message.size() );
            message.insert( message.begin(), header.begin(), header.end() );
            return message;
        }
```

Listing 12: AsterixCategory000.cpp