FMT (FLIGHT SOFTWARE MEMORY TRACKER) FOR CASSINI SPACECRAFT' - SOFTWARE ENGINEERING USING JAVA

Edwin P. Kan, Hal Uffelman, and Allan H. Wax
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, Ca. 91109

ABSTRACT

The software engineering design of the Flight Software Memory Tracker (FMT) "Tool" is discussed in this paper. FMT is a ground analysis software set, consisting of "utilities" and "procedures", designed to track the flight software, i.e., "images" of memory load and updatable parameters of the computers on-board the Cassini spacecraft. FMT is implemented in Java.

1. INTRODUCTION

Tracking flight software (FSW) images on spacecraft is a vital activity in ground systems and mission operations. Particularly true for one-of-a-kind spacecraft built for space and planetary exploration, FSW requires constant maintenance, patches, parameter changes, and occasionally even complete new memory loads. Complete and accurate knowledge of current and past FSW images is essential.

In the history of spacecraft operations at the Jet Propulsion Laboratory (JPL), ground mission operations analysts utilize various degrees of automation, integration of software tools and manual procedures to track FSW. While dynamic memory addresses can only be tracked by full-up hardware and software simulation, static memory addresses, constants, and certain quasi-static **parameter addresses are always tracked. For such address spaces of interest, an up-to-date FSW** image, a FSW image at a specific time in history, and a trend of certain parameters over time, are often the basis for analysis, diagnosis and prognosis.

The FSW Memory Tracker (FMT) is a ground systems "Tool", designed expressly for tracking the FSW code and parameter address spaces of interest, FMT maintains a history of every FSW image copy on-board the spacecraft. These images are living images, which are updated when the on-board images are updated, can be "evaluated" at any specific time of history, can be queried for history and statistics, and can be processed to produce human readable parameter values in engineering units instead of machine representation, etc. While on-board FSW can be updated by uplink commands transmitted to the spacecraft, the same commands can be parsed and interpreted by FMT into update data groups appended to FMT images. Memory readout of spacecraft address spaces (in addition to normal telemetry downlink) can also be used to update FMT images. Subsequent to intended or unintended reinitialization, or in order to revive specific images / parameters at specific instances of history, generic commands to update on-board FSW images and FMT images can also be generated by FMT through an analysis of differences in specific FMT images.

The software design of FMT has to pay attention to the maintenance of multiple FSW images at many instances of time. For Cassini, while each image is necessarily a 5 12K word (16-bit per word) image, this task is by no means unmanageable. The challenge is in maintaining multiple time copies of multiple physical copies of the images.

The approach and solution to such a large data handling challenge is achieved in FMT by the use of novel, yet intuitive, data and file structures. While the processing of data in FMT is fairly straightforward (the command parsing and updateByCmd requires an interface to a long-established JPL program set), the software engineering of FMT requires multiple but similar classes of algorithms for different data structures. As such, the property inheritance feature (among other attractive features) of an object oriented programming language such as C++ or Java can be utilized to full advantage.

Using the idea of "procedures" and "utilities", FMT is implemented as a program set, consisting of multiple "utility" programs, some of which are programs that also call other FMT utilities. FMT users are best served by executing scripts, such as UNIX scripts, to achieve the higher-level objectives of FMT.

FMT was developed as part of a the Multimission Spacecraft Analysis System (MSAS), under the auspcies of the Jet Propulsion Laboratory Multimission Ground Systems Office. Their extensive hardware configuration and software capabilities can be found in Ref. 1 and 2. The functional design and software specification of FMT can be found in Ref. 3.

FMT is implemented in Java with the exception of certain programs with which FMT interfaces. While the application and operation aspects of FMT will be discussed in a different paper (Ref. 4), its software design aspects are discussed in this paper.

2. FMT FOR CASSINI GROUND SYSTEMS

The Cassini spacecraft is scheduled for launch in October, 1997, will spend seven years of interplanetary flight, to arrive and be inserted into orbits around planet Saturn and its moons. Major scientific goals of Cassini after orbit insertion include remote science data collection, during a primary mission which ends in year 2008.

During its twelve years of primary mission, the Cassini software is expected to be changed, patched, reinitialized, reloaded, and have parameters updated. These changes will mostly be commanded by ground mission operations, and occasionally be induced on-board by unforseen faults and fault responses.

On Cassini, apart from the science instruments which all have certain computing and sequencing capabilities, there are two major spacecraft subsystems that have extensively computing facilities, namely the AACS (Attitude and Articulation Control Subsystem) and the CDS (Command and Data Subsystem). Both AACS and CDS are equipped with dual redundantMIL-STD-1750A computers with 5 12K memory each. In addition, there are two redundant solid state recorders (SSR), each with a capacity of 2 gigabits each. On each SSR are resident multiple (4) copies of AACS and CDS FSW memory load images.

Counting these FSW images, AACS tallies 10 copies (there are another two additional copies on extended memory). CDS tallies 10 copies. The SSR copies are formatted basically like the compiled load images, with additional checksums appended to the data records. The actual RAM images are 5 12K-word images, basically address-value pairs.

All SSR data records are tracked by FMT. On-board SSR records can be changed by the ground via special commands, comprising ALF, ALF_SKIP and ALF_END commands (Ref. 4).

Not all 5 12K RAM addresses are, or can be, tracked by FMT. The static code addresses and constants comprise about 50% of the 5 12K addresses. Of interest to mission operations analysts are parameter addresses such as tables, mathematical constants, spacecraft properties, and algorithm multipliers, etc. "Variables" that tend to be altered by FSW during nominal FSW and sequence execution are normally included in spacecraft telemetry downlink, and can be subjected to a different tracking outside of FMT. For FMT, the special class of parameter lists and data values tallies several hundred or more records. This set of parameters are populated with values upon software initialization. They may be altered via special parameter change commands or brute force memory_write (MEM_WRITE and PATCH) commands (Ref. 4).

FMT is intended for use by Cassini AACS and CDS mission operations analysts. Sharing the design pedigree of analysis tools developed for Mars Obversver and Galileo (Ref. 6 and 7), FMT is a software tool to be used daily or weekly by analysts to maintain and update spacecraft FSW images. FMT activities include a one-time data initialization and setup; updating of images caused by uplink commands; updating of images as reflected by downlink inputs; generation of difference images and uplink commands (stems and parameters); and miscellaneous analysis such as history, statistics and queries. The frequency of "official" FMT execution depends on the frequency of uplink and downlink activities, which in turn depends on the availability of Deep Space Network coverage. Otherwise, FMT can be exercised as an end-to-end analysis, synthesis, and predict tool.

For Cassini, FMT interfaces with a JPL uplink software tool, called SEQ_GEN, which is used for the parsing of uplink commands.

While FMT is now customized to Cassini, it is applicable to the class of spacecraft designed and operated by JPL. Upon transcending design features specific to JPL command and telemetry systems, the data file structure design and processing methodology of FMT can be extended to general spacecraft systems.

3. FMT SOFTWARE ARCHITECTURE AND DATA STRUCTURE

The accomplishment of the objectives of FMT is done via the concept of multiple FMT utilities scripted together to achieve a function or a class of functions. Due to the multiplicity of images, over spatial and temporal span, a not-so-large data tracking and database task can only be solved **via well conceived system-engineered data structures and processing architecture. ("Spatial" refers** to multiple copies over multiple subsystems. "Temporal" refers to updates of FMT images, where updates could be as frequent as seconds or minutes, namely the time interval between update commands.)

The FMT software and data structure design utilizes the concept of maintaining temporal images using update deltas. Where and when required, "evaluation" and "refresh" operations can be performed on these composite "images" for time-specific "peeking" and "poking".

Figure 1 shows the context diagram and Figure 2 shows the (Level 1) Data Flow Diagram of FMT.

Evident in the Data Flow Diagram are the various data file types including:

.alv	.ealv	.xalv	(SSR "Assisted_Load_File" data classes)
.fmt	.efmt	.xfmt	(RAM FMT data classes)
.amf	.eamf		(Attribute_Model_File classes)
.adb	.eadb		(Attribute_Data_Base classes)
.msk	.emsk		(Data Mask classes)

The utility of, hence class extension and property inheritance between, these data structures can be easily exemplified by the following:

At load time, the RAM .fmt file contains the data records pertaining to load time, i.e., one data group. Each group has a data group header comprised of time (SCET) and image type (RAM I SSR I etc.), and $$EOG (an indicator for "end-of-group"). Each data record is structured to show RAM start_address, data values of up to sixteen RAM addresses and a new line character. The .fmt file has a file header record and $$EOF (an indicator for "end-of-file").

When an update takes place, the update is implemented in the **.fmt** file with a new time-stamped data group appended to the previous data group(s). With such appending of data group(s), the **.fmt** file takes the nomenclature of **.efmt** ("**enhanced**" fret) file.

For certain update operations, delta update files with multiple data groups are generated by FMT utilities. Due to the "spatial" nature of FMT, these update groups need to be designated to specific **"spatial" images, e.g.** AACS_A computer vs **AACS_B** computer. Hence, the nomenclature of a **.xfmt** ("extended fret) file, where the data group header takes an extra descriptor (a Software Image Designator, MD), e.g. **SCET;** RAM; **AACS_A**. The following examples show the skeleton of a **.xfmt** file:

```
DATA_FI LE_HEADER
1997 -070 TOO:00:00.000; RAM; AACS_A
000670 e511 e522 740a 8a0f 0000 7b04 8a00 1df5 . . . 1df2 85ff 0003 7ff0 7ef0
000680 23e8 7ff0 7ef0 2384 7ff0 7ef0 2309 7ff0 . . . 7ef0 24a8 7ff0 7ef0 24af
$$EOG
1997 -298 T12:00:01 .001; RAM; AACS_A, AACS_B
038e80 4189 37f7
039366 7dce 000d 0000
$$EOG
$$$EOF
```

All twelve file types stated above have identical data header, data group, **.ttt** and **.ettt** file structure. Data record structures for different data types are different. A RAM fmt data record has been shown as a 17-element record, consisting of an address followed by sixteen 4-nibble hex values. As another example, an **.adb** data record takes the following form (in single line):

```
038e80,0, A5.6.7,4189 37 f7,1997-098T00: 00:00.000,0 .001,4/8/1997 0:00:00,
      ACL_PARAMETERS, Att itude_Deadband, 6, posDB [ 0 ] ,2, f lost, 7DEADBAND, 1, 0.001,
      rad,0 . 0005,0.35,,,,,,,,,,,
```

Of note is that **all** these files are ASCII text files, which can be imported and exported to popular COTS (commercial off-the-shelf) personal computer editor and application programs, hence making it relatively painless to perform front-end and tail-end processing. FMT is written in Java, which means that its **bytecodes** are independent of, and hence executable on multiple end-user computing platforms.

4. FMT SOFTWARE DESIGN USING JAVA

At the time of this writing, FMT comprises 124 classes of objects. Within these 124 classes are similar groups of classes, befitting the commonality of data file structures designed into FMT. For illustration, the following are the formative classes defined for **.fmt** data:

FMT.java	**FMTGroup.java**	**FMTGroupData.j** ava
FMTGroupDataElement.java	**FMTError.java**	FMTGroupError,j ava
FMTGroupDataElementError.java		
GroupHeader.java	**GroupData.java**	
GroupHeaderError.java	GroupDataElementError.java	

The class hierarchy for this group of classes is illustrated in Figure 3.

The FMT design embodies many other Java programming language features. The garbage collection feature is used extensively in FMT codes. This feature is particularly important because FMT data files, notably **.efmt** files, can be very large. Each time when a **.efmt** file is processed, thousands to tens of thousands of objects (each FMTGroupDataElement is an object) are generated, dynamically space allocated, and marked for deletion from memory as the individual object has been processed. In this way, the programmer is freed of the underlying task of managing space, which otherwise requires complex coding logic and a great deal of debugging effort.

The Java feature of interfaces and "child" classes are used extensively, e.g., the Java keywords "import", "implement", and "extend". FMT, however, chooses to use one single package since all fmt objects have structures in common, and individual programs exist under a single logical umbrella. Three of the four Ps, as discussed in Ref. 6, namely protection schemes using "public", "protected", and "private" are used all through FMT. (The fourth P, namely package protection, is not exercised in FMT.)

Java exception handling features using keywords "final", "throw" and "try" are used for catching unusual conditions, detecting semantic and syntatic errors in the data, and checking bounds. Particularly, arrays in Java are inherently checked for bounds at runtime; thus relieving the programmer of the difficult task of detecting invalid array reference due to data values in the processing .efmt files (and other files). In each of the above exception handling schemes, code to handle these cases is written once and normal program flow assumes the non-error case.

Java is bundled with an extensive library, including java.lang, java.util, and java.io. Heavily used in FMT are the Vector and Properties operations, No additional libraries, whether from other commercial sources or private individual sources, were needed. The task of certifying FMT becomes so much easier - when the embedded Java libraries are considered a priori certified.

The current FMT design has 23 programs, or "utilities", listed as follows:

fmtbeheadeadb	fmtcmdstemgen	fmtconvert	fmtcreate
fmtcs16	fmtdiff	fmtdv2eu	fmteu2dv
fmtextractdes	fmtfilteralv	fmtmaskalv	fmtmasktdc
fmtmaskxfm	fmtmemupdatecmdgen	fmtmerge	fmtmro
fmtquery	fmtrefresheadb	fmtreinit	fmtsort
fmtuser	fmtupdatebycommand	fmtxemerge	

These "utilities" are normally executed in scripts, i.e. "procedures", in order to achieve an overall high level objective of fret, as discussed in Section 2, The scripting of these utilities will be discussed in details in another paper (ref. 7).

5. SUMMARY

The Flight Software Memory Tracker (FMT) is being embraced as a new and powerful ground analysis tool for the Cassini spacecraft mission. The overall objectives of FMT and its software architecture / design/Java implementaiton have been discussed in this paper.

This Java implementation of FMT achieves three of the four S's discussed in Ref.6, namely, small, simple and safe (the fourth S is secure, relating to over the internet security, which is not pertinent in the present context). The Java implementation of FMT code will permit the code to be reused for different projects, which may adopt different software architecture, and which may have multiple users with various kinds of computing platforms. The use of Java offers the opportunity to obviate multiple developments of the same application. The same code will run on computers regardless of their architecture and operating system, as long as a Java virtual machine exists on that computer; that being a definite trend into the next century.

Acknowledgement
This work was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under contract to the National Aeronautics and Space Administration. The functional requirements of this work were developed by R. Morillo; engineering design/ software specification by E. Kan and H. Uffelman; software design by A. Wax; and implementation by A. Wax, J. Tusynzski and T. White.

References
1. Hill, M. (custodian), "Multimission Spacecraft Analysis System - Functional Requriements Document," Jet Propulsion Laboratory, Document #JPL D-9173,Rev.D, July 10, 1996.
2. Murphy, S. C., et.al., "Customizing the JPL Multimission Ground Data System," Proc. SPACEOPS 1994, 3rd Int. Symp. on Space Mission Operations and Ground Data Systems, held at GSFC, Greenbelt, Md., USA, Nov. 14-18, 1994.
3. Kan, E. P., and H. Uffelman, "CDS (Command and Data Handling Subsystem) Package Requirements Document - Flight Software Memory Tracker," Jet Propulsion Laboratory Document #JPL D-9173 (Section 3.2), Dec. 1,1997.
4. Tapia, E. (custodian), "Cassini Functional Requirements 3-291, Uplink Formats & Command Tables," Jet Propulsion Laboratory Document #CAS-3-291, Rev. E, Jan, 24, 1997.
5. Kan, E. P., "Mission Operations Data Analysis Tools for Mars Observer Guidance and Control," Proc. SPACEOPS 1994, 3rd Int. Symp. on Space Mission Operations and Ground Data Systems, held 'at GSFC, Greenbelt, Md., USA, Nov .14- 18, 1994.
6. Kan., E.P., "Low Bit Rate Autonomous Spacecraft - End-to-End G&C System Design," Proceeding of the AIAA GNC Conf., (American Institute of Aeronautics and Astronautics, Guidance and Control Conference) Paper #96-3925, San Diego, CA 7/23 -3 1/96.
7. Kan, E. P. et al., "Tracking Flight Software in Cassini Mission Operations Using the FMT Tool," (submitted to) SPACEOPS 1998, 5th Int. Symp. on Space Mission Operations and Ground Data Systems, to be held at Tokyo, Japan, 6/1-5/98,
8. Lemay, L. and C. Perkins, "Teach Yourself Java 1.1 in 21 Days," Sams.net Publishing, 1997.

Figure 1. **Flight_Software_Memory_Tracker (FMT)** Context Diagram (Level O)

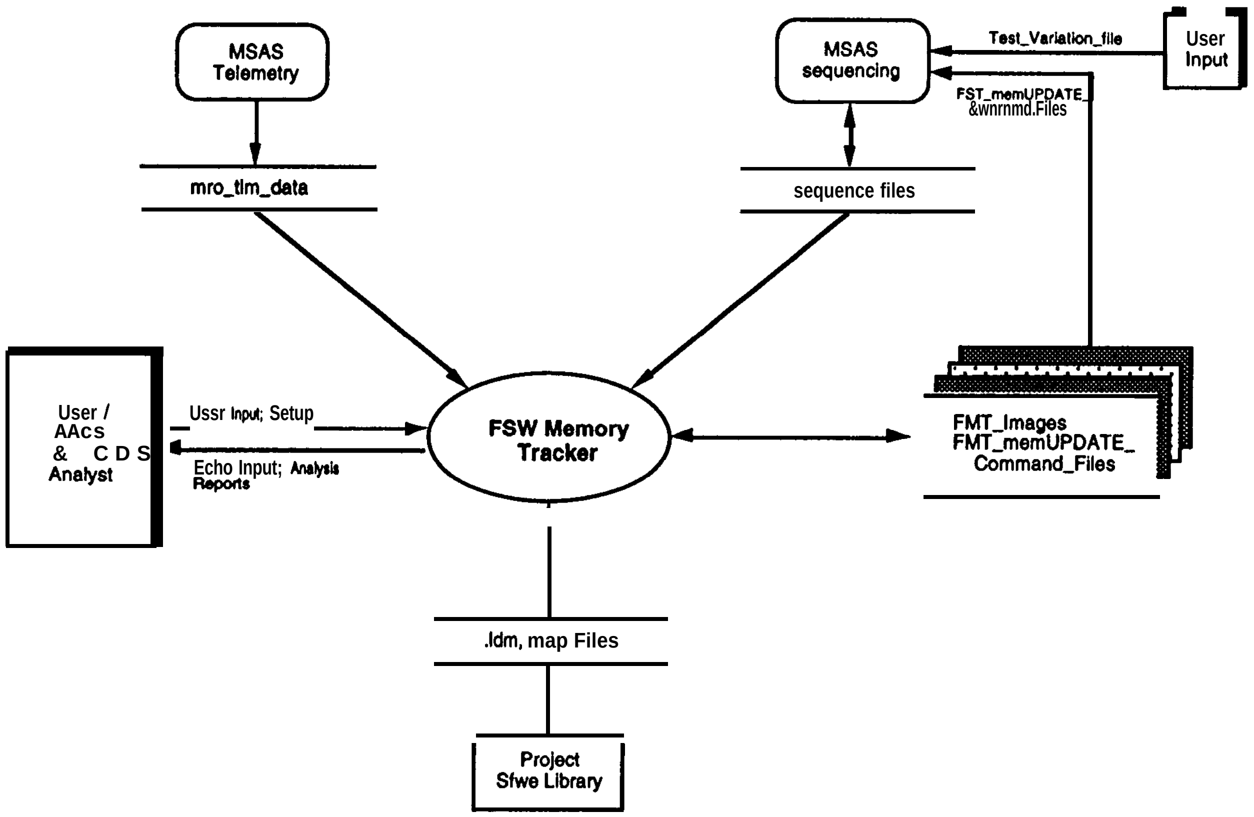


Figure 2. FSW_Memory_Tracker **(FMT)** DFD (Level 1)

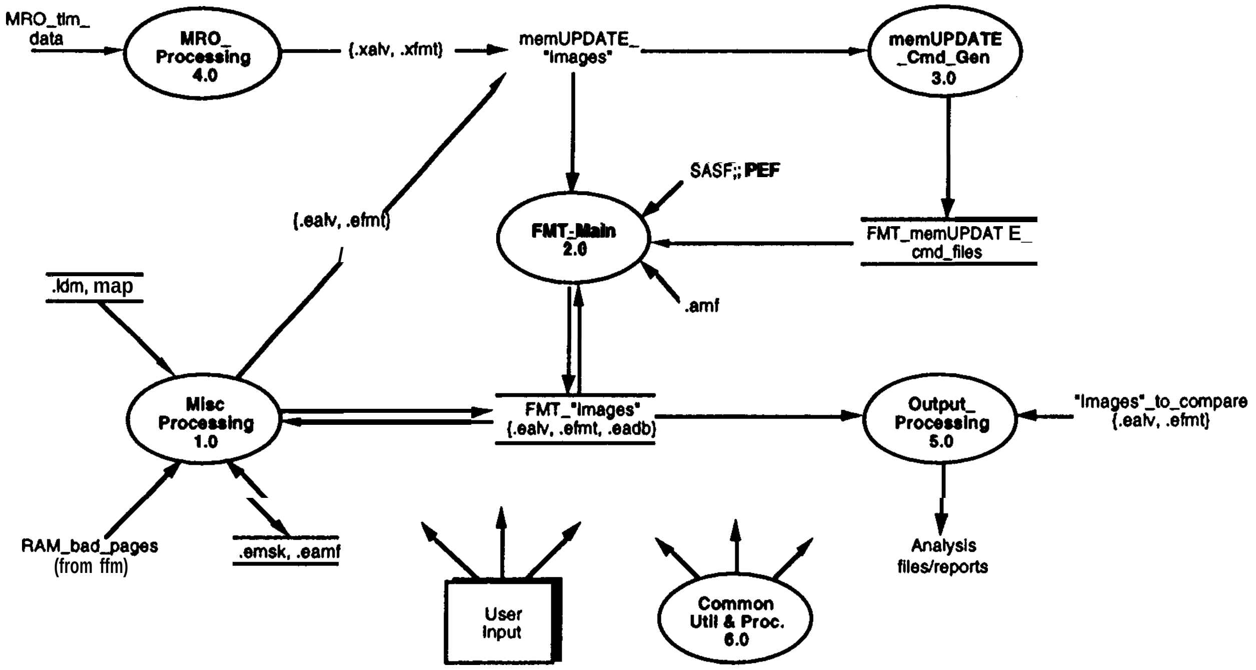
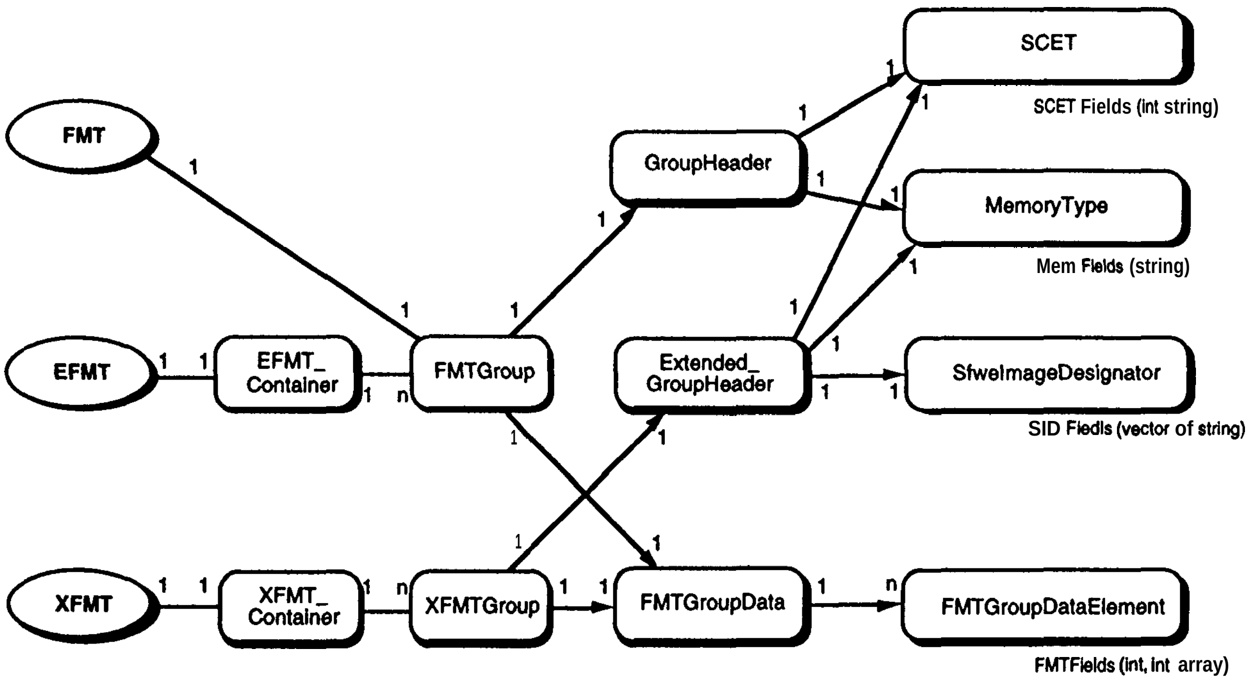


Figure 3. Class Hierarchy Diagram for FMT, **EFMT,** and XFMT Java Classes


FMT
EFMT
XFMT
EFMT_ Container
XFMT_ Container
FMTGroup
XFMTGroup
GroupHeader
Extended_ GroupHeader
FMTGroupData
SCET
SCET Fields (int string)
MemoryType
Mem Fields (string)
SfwelmageDesignator
SID Fiedls (vector of string)
FMTGroupDataElement
FMTFields (int, int array)
1
n