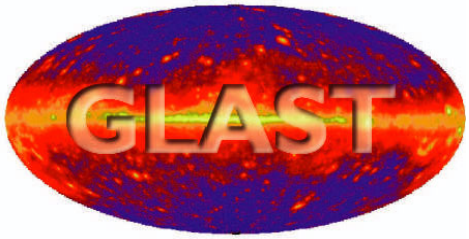


LAT Flight Software

Attitude Processing

Type:	Design Description
Version:	V1-3-0
Author:	D.L. Wood
Created:	20 May 2003
Updated:	11 May 2007
Printed:	18 May 2007

A description of the LAT flight software attitude processing package.

Contents

0 Introduction 2
0.0 Requirements 2
0.1 Reference Documents 2
1 Coordinate Systems 3
1.0 J2000 Coordinate System 3
1.1 GLAST SC Coordinate System 3
1.2 LAT Instrument Coordinate System 4
1.3 Vectors 4
1.4 Attitude Quaternions 4
2 Software Interface 6
2.0 Control Functions 6
2.0.0 ATT_get() 6
2.0.1 ATT_init() 6
2.0.2 ATT_update() 7
2.1 Transform Functions 7
2.1.0 ATT_xform() 7
2.2 Software Examples 7
3 Diagnostics and Errors 10
4 Simulation Support 12
4.0 Attitude Simulation File Format 12
5 Unit Testing 16
5.0 Unit Test Coverage 16
5.0.0 Constituent: att 16
5.1 Running the Unit Test 16

Figures

Figure 1 - Attitude Simulation File Format 12
Figure 2 - Attitude Simulation Contribution Format 13
Figure 3 - SC Attitude Data Record 14
Figure 4 - SC Position Data Record 14

Tables

Table 1 – Attitude Processing MSG Codes 10

0 Introduction

The LAT flight software uses the attitude processing package ATT to perform transforms between the local LAT instrument coordinate system and the J2000 inertial coordinate system. Such transforms are necessary to perform some of the flight software on board science processing.

0.0 Requirements

The LAT on-board attitude processing software shall perform the following:

1. Transform a vector in LAT instrument body coordinates into a vector in J2000 coordinates for a given time within the attitude time history limit.
2. Accept as input the SC-J2000 four element quaternion, the SC three element angular velocity vector, and the timestamp for the data.
3. Maintain an attitude time history for the SC for up to 32 (TBR) seconds.
4. Allow the alignment error between the SC and LAT to be set as a configuration parameter.

0.1 Reference Documents

GLAST Mission System Specification, Rev A, NASA GSFC, Oct 2000.

LAT Coordinate System (LAT-TD-00035-01), S.Ritz, Nov 2000.

Spacecraft Attitude Determination and Control, Ed. James R. Wertz, 1978.

MSG User Manual, LAT Flight Software User Manual.

1 Coordinate Systems

This section outlines the various coordinate systems recognized by the attitude processing software.

1.0 J2000 Coordinate System

The J2000 coordinate system has the origin at the center of the Earth. The *z* axis is aligned with the Earth's axis of rotation. The +X axis points to a fixed location on the celestial sphere. The X-Y plane is aligned with the plane of the Earth's equator.

The LAT flight software is required to report science transient events in J2000 coordinates. In this case, the spherical form of coordinates is used, discarding the *r* coordinate, or length of the vector. The remaining angular coordinates are reported as (RA,DEC). The RA coordinate ranges from 0 degrees to 360 degrees. The DEC coordinate ranges from +90 degrees to –90 degrees. The following Cartesian vectors (*x, y, z*) and spherical vectors (RA, DEC) are equivalent:

```
(1,  0,  0)        =     (0,   0)
(0,  1,  0)        =     (90,  0)
(-1, 0,  0)        =     (180, 0)
(0, -1,  0)        =     (270, 0)
(0,  0,  1)        =     (0,  90)
(0,  0, -1)        =     (0, -90)
```

1.1 GLAST SC Coordinate System

The GLAST spacecraft (SC) coordinate system is defined in the mission specification as the observatory body fixed coordinate system. The Z axis is collinear with the geometrical centerline of the observatory. The +Z axis points in the direction of the LAT instrument field of view. The Y axis is parallel to the solar array drive axis.

The spacecraft provides the LAT instrument with a periodic attitude message containing a four element quaternion. The quaternion provides an instantaneous representation of the attitude of the SC coordinate system with respect to the J2000 coordinate system. This attitude quaternion is the key to the LAT flight software attitude processing.

For the purposes of LAT flight software attitude processing, the approximation is used that the origin of the SC coordinate system is coincident with the origin of the J2000 coordinate system when calculating celestial pointing directions.

1.2 LAT Instrument Coordinate System

The LAT instrument coordinate system is defined such that the coordinate axes are parallel to the corresponding SC coordinate axes. That is, the +Z axis points in the direction of the instrument field of view and the Y axis is parallel to the solar drive array axis. For the purposes of LAT flight software, the approximation is made that the origin of the LAT coordinate system is coincident with the origin of the SC coordinate system, and therefore, the J2000 coordinate system. To allow for alignment errors, however, the LAT flight software will assume that the LAT and SC coordinate systems may be rotated with respect to each other. This alignment correction rotation will be configurable by an on-board file. This file will represent the alignment correction rotation as a single quaternion.

1.3 Vectors

For the most part, the LAT flight software only considers the directions of vectors, not their magnitudes. To simplify calculations, internally, the attitude processing software works with unit vectors.

In the LAT coordinate system, vectors such as particle track directions are represented in Cartesian form (x2, y1, z1). The primary function of the flight software attitude processing is to take such a vector and return a unit vector (x2, y2, z2) which represents the original vector in the J2000 coordinate system. As a convenience, the attitude software will provide a method to transform the (x2, y2, z2) vector into reduced spherical form (RA, DEC), discarding the magnitude.

Some useful properties of unit vectors:

1. The angle between two unit vectors V and U is arccos(V • U)

1.4 Attitude Quaternions

Quaternions are a method of representing the attitude of one coordinate system with respect to another. Quaternions contain four elements, represented either as (i, j, k, s) or (q1, q2, q3, q4), with the first three elements the vector and the last the scalar.

Some useful properties of quaternions:

1. The conjugate of a quaternion Q = (q1, q2, q3, q4) is Q′ = (-q1, -q2, -q3, q4)

2. The product C = (qc1, qc2, qc3, qc4) of two quaternions A = (qa1, qa2, qa3, qa4) and B = (qb1, qb2, qb3, qb4) is defined as C = A * B, where:

$$qc1 = (qa2 * qb3) - (qa3 * qb2) + (qa4 * qb1) + (qa1 * qb4)$$
$$qc2 = -(qa1 * qb3) + (qa4 * qb2) + (qa3 * qb1) + (qa2 * qb4)$$
$$qc3 = (qa4 * qb3) + (qa1 * qb2) - (qa2 * qb1) + (qa3 * qb4)$$
$$qc4 = -(qa3 * qb3) - (qa2 * qb2) - (qa1 * qb1) + (qa4 * qb4)$$

   Quaternion multiplication is not commutative.

3. The identity quaternion is I = (0, 0, 0, 1). For any quaternion Q, Q * I = Q.
4. If Q′ is the conjugate of a quaternion Q, then Q * Q′ = I
5. If a quaternion Q represents the attitude between coordinate systems, and V = (x1, y1, z1) is a vector in one coordinate system, then:

$$U = Q' * V' * Q$$

Here, the vector V is transformed into quaternion V′ = (x1, y1, z1, 0) and Q′ is the conjugate of Q. The resulting quaternion U = (q1, q2, q3, q4) can then be used to get the vector U = (q1, q2, q3), which is the representation of vector V in the second coordinate system.

The examples below show some quaternion values corresponding to simple, common rotations.

(0, 0, 0, 1)	Identity quaternion, no rotation
(1, 0, 0, 0)	180' turn around X axis
(0, 1, 0, 0)	180' turn around Y axis
(0, 0, 1, 0)	180' turn around Z axis
(sqrt(0.5), 0, 0, sqrt(0.5))	90' rotation around X axis
(0, sqrt(0.5), 0 sqrt(0.5))	90' rotation around Y axis
(0, 0, sqrt(0.5), sqrt(0.5))	90' rotation around Z axis
(-sqrt(0.5) , 0, 0 sqrt(0.5))	-90' rotation around X axis
(0, -sqrt(0.5), 0 sqrt(0.5))	-90' rotation around Y axis
(0 , 0, -sqrt(0.5), sqrt(0.5))	-90' rotation around Z axis

2 Software Interface

This section briefly describes the attitude processing software interface.

2.0 Control Functions

2.0.0 ATT_get()

```
ATT_Control *ATT_get(void)
```

The attitude processing software maintains a global control and state structure. This function returns a pointer to this global structure. The functions in this library take this value as their first parameter. The function *ATT_init()* must be called at startup before using the value returned by this function.

2.0.1 ATT_init()

```
unsigned int ATT_init(ATT_Contol *ctl, unsigned int numSecs)
```

This function initializes the global attitude processing control structure. The *numSecs* parameter is the number of seconds to maintain the SC attitude history list. The memory for the history list is allocated here.

2.0.2 ATT_update()

```
unsigned int ATT_update(ATT_Control *ctl, const WCT_time t, const
      double *Q, const float *W)
```

This function accepts a new update of SC attitude information. The new information is added to the head of the attitude history list and any stale information in the attitude history list is discarded. It is expected that this function is called in the SC telecommand processing task. The *Q* parameter points to an array of four elements where each element is one component of the SC-J2000 quaternion *(q1, q2, q3, q4)*. The *W* parameter points to an array of three elements where each element is one component of the SC angular velocity vector *(wx, wy, wz)*. The *t* parameter is the timestamp for the attitude information. Before storing the attitude quaternion delivered by the SC, *ATT_update()* will first multiply this quaternion by the configurable alignment correction quaternion.

2.1 Transform Functions

2.1.0 ATT_xform()

```
unsigned int ATT_xform(const WCT_time t, const double *Vl, double *Vj)
```

This function accepts as input a vector in LAT instrument coordinates and produces the same vector in J2000 coordinates as output. The *Vl* parameter is a pointer to an array of three elements where each element is one component of the vector in LAT coordinates *(lx, ly, lx)*. The *Vj* parameter is a pointer to an array of three elements. Upon successful return, the ATT_xform() function will store the three components of the vector in J2000 coordinates *(jx, jy, jz)*. The *t* parameter is the timestamp of the vector component information. This value is used to lookup the SC attitude information for that time.

2.2 Software Examples

The first example shows how to call the ATT_update() function when a new SC attitude message has arrived.

```
#include “ATT/ATT.h”
#include “MSG/MSG_pubdefs.h”
#include “PBS/WCT.h”

WCT_time ltime;
WCT_time_sat *stime;
ATT_Control *ctl;

/* get a reference to the attitude global handle */

ctl = ATT_get();

/* convert SC time format to LAT WCT time format */

stime = (WCT_time_sat*) &my_msg->sec;
ltime = WCT_from_sat(*stime);

/* inform attitude history list of update */

status = ATT_update(ctl, ltime, my_msg->quat, msg->msg_avel);
if(_msg_success(status) == 0)
{
      /* error handler */
}
```

The second software example shows the use of the vector transform functions.

```
#include “ATT/ATT.h”
#include “MSG/MSG_pubdefs.h”


double latVector[3];
double j2000Vector[3];
double sphrVector[2];
ATT_Control *ctl;

/* get a reference to the attitude global handle */

ctl = ATT_get();

/* convert LAT Cartesian vector to J2000 Cartesian vector */

status = ATT_xform(ctl, my_time, latVector, j2000Vector);
if(_msg_success(status) == 0)
{
      /* error handler */
}

/* tranform J2000 vector to RA,DEC coordinates */

ATT_vec_cart_to_sphr_deg(j2000Vector, sphrVector);
```

3 Diagnostics and Errors

The *att* constituent supports the *MSG* status reporting system. The following message codes are defined.

Table 1 – Attitude Processing MSG Codes

MSG Facility	MSG Code	Description	Parameters
ATT	ATT_SUCCESS	Success.	None.
	ATT_EFPARAM	A function call parameter value is out of range.	The name of the bad parameter.
	ATT_EMEMALOC	Memory allocation failure.	The name of the memory object.
	ATT_EOBJINIT	Object initialization failure.	The name of the object.
	ATT_ETIMERNG	Time value requested in lookup is not within attitude history list.	The time requested and the closest time found in the history list.
	ATT_ETIMEORD	Time value for update is out of order.	The update time and the most recent time in the history list.
	ATT_ESTATE	Attitude history list bad state.	A brief description of the condition.
	ATT_IRESET	Attitude history list has been reset.	None.
	ATT_IUPDATE	Attitude history list update report (normally not issued).	The update information.
	ATT_ILOOKUP	Attitude history list successful lookup (normally not issued).	The information found in the lookup.

	ATT_IHISTORY	Attitude history list lookup attempt(normally not issued).	The time of the history list element.

The *att* constituent signals all error messages at run time.

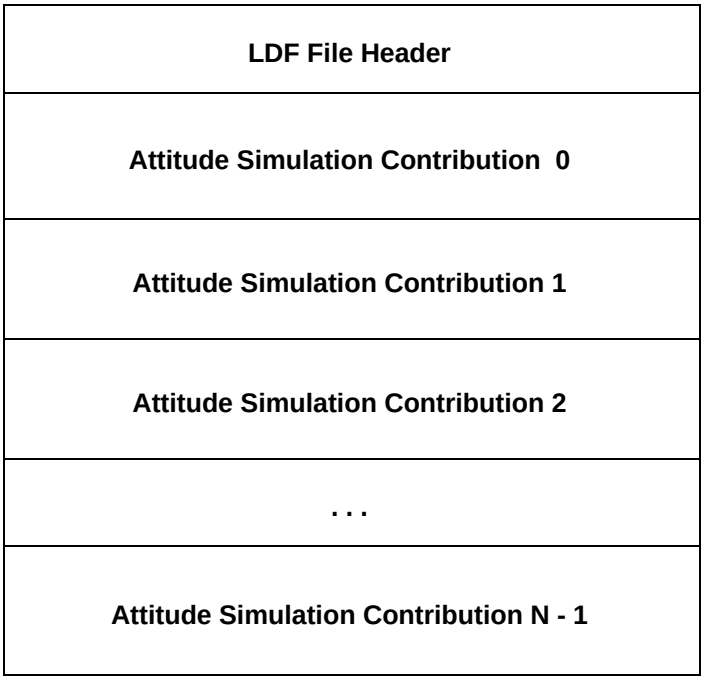
4 Simulation Support

4.0 Attitude Simulation File Format

This section proposes a binary file format for simulating the SC attitude and position information.

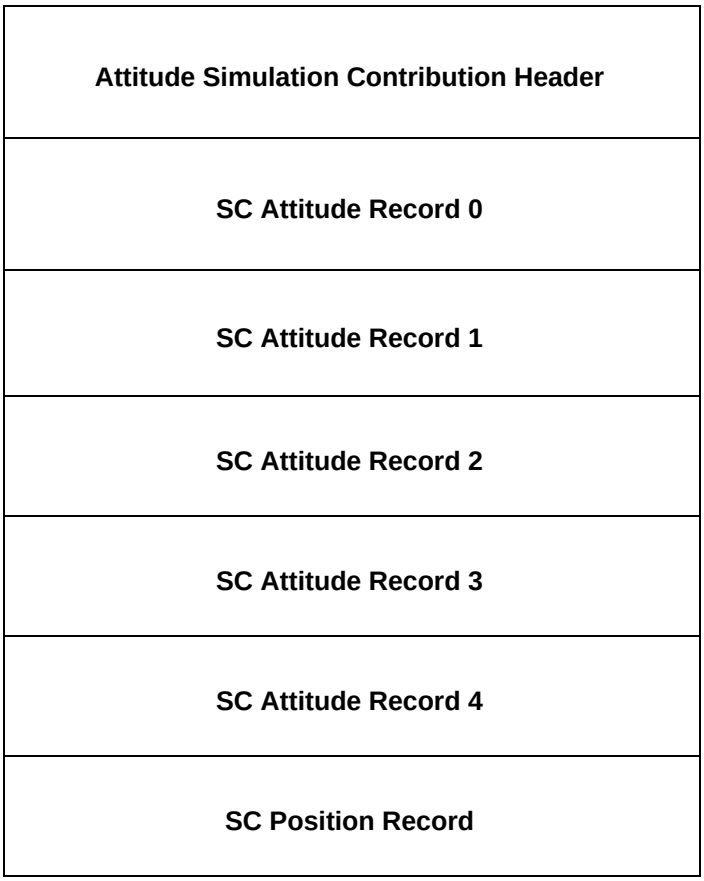
The simulation binary file general formation is show below.

Figure 1 - Attitude Simulation File Format



The attitude simulation files use the stanadard LDF header. The LDF header type value is 0xF0002, and the LDF header version value is 0x000. Because the contributions are always of the same type and size, the LDF header length member provides an easy way to derive the number of contributions contained within one simulation file. Each contribution within the LDF file has the format shown in Figure 2. The attitude simulation contributions always contain exactly five SC attitude simulation records, followed by excactly one SC position simulation record. The contribution headers have ID values of 0xFFFFFFFF (TBR).

Figure 2 - Attitude Simulation Contribution Format



Each SC Attitude Data record has the format shown in Figure 3.

Figure 3 - SC Attitude Data Record

Record Timestamp (64-bits, in SC [seconds, mircoseconds] format)
SC-J2000 Attitude Quaternion Element q1 (i) (64-bit IEEE)
SC-J2000 Attitude Quaternion Element q2 (j) (64-bit IEEE)
SC-J2000 Attitude Quaternion Element (k) q3 (64-bit IEEE)
SC-J2000 Attitude Quaternion Element q4 (s) (64-bit IEEE)
SC Angular Velocity X Component (rad/sec) (32-bit IEEE)
SC Angular Velocity Y Component (rad/sec) (32-bit IEEE)
SC Angular Velocity Z Component (rad/sec) (32-bit IEEE)

Each SC Position Data Record has the format shown in Figure 4.

Figure 4 - SC Position Data Record

Record Timestamp (64-bits, in SC [seconds, mircoseconds] format)
SC Position X Component (m) (32-bit IEEE)
SC Position Y Component (m) (32-bit IEEE)
SC Position Z Component (m) (32-bit IEEE)

SC Position Velocity X Component (m/sec) (32-bit IEEE)
SC Position Velocity Y Component (m/sec) (32-bit IEEE)
SC Position Velocity Z Component (m/sec) (32-bit IEEE)

5 Unit Testing

The *ATT* package provides a unit test which test the libraries both for successful baseline functionality as well as some error coverage paths.

5.0 Unit Test Coverage

5.0.0 Constituent: att

The basis of the att constituent unit test is a set of data sets derived from an STK simulation. The data sets cover basic SC attitudes for arbitrary times. In addition to outputting time, quaternion, and angular velocity information, the data sets include test vectors and the the components of those test vectors in both LAT and J2000 coordinates. The attitude history list is initialized, and various selections from the data sets are loaded into the history list by calling ATT_update(). The ATT_xform() function is then called, using the test vector LAT components. The resulting J2000 vector components are compared against the values calculated by STK.

5.1 Running the Unit Test

The *ATT* package unit test may be run directly on UNIX hosts by typing *att_unit_test* at the command shell. If the proper modules are loaded, then the unit test may be run on VxWorks hosts by typing *att_unit_test* at the VxWorks shell prompt. The preferred method, however, is to use LTX. As part of the *ATT* package, a LTX script is provided which defines a test called *att_unit_test*.