

The CDGenericPlatform Class

Scott D. Brown
Digital Imaging and Remote Sensing Laboratory
Rochester Institute of Technology
Rochester, NY 14623

March 9, 2006

Contents

1	Copyright Notice	1
2	Overview	1
2.1	Document Version	4
3	The CDGenericPlatform class	5
3.1	Parent Class	5
3.2	Class Dependancies	5
3.3	Class Implementation	5
3.4	Public Type Definitions	6
3.4.1	The orientation angle reference	6
3.5	Private Variable Declarations	6
3.5.1	The orientation angle reference type	6
3.5.2	Rotation order	6
3.5.3	The list of known positions/orientations	7
3.5.4	The cache of computed positions/orientations	7
3.5.5	The uncertainty parameters	7
3.5.6	The average altitude	8
3.5.7	The average orientation angles	8
3.6	Constructors	8
3.6.1	Default Constructor	8
3.7	Destructor	9

3.8	Accessors and Mutators	9
3.8.1	Set and get the orientation type	9
3.8.2	Clear the current data entries	10
3.8.3	Set and get the uncertainty enable flag	10
3.8.4	Add a data entry	11
3.8.5	Get number of position entries	11
3.8.6	Get average orientation angles	11
3.8.7	Get average orientation angles	11
3.8.8	Initialization function	12
3.8.9	Get the position data at a specific index	15
3.8.10	Get the position for a specific time	15
3.8.11	Compute the full transform [private]	18
3.8.12	Load data from an XML file	20
3.8.13	Load data from XML document	20
3.8.14	Save data to an XML file	26
3.8.15	Save data to an XML document	28
4	Header file	32
5	Source file	33
6	Revision History	34

1 Copyright Notice

Copyright Rochester Institute of Technology, Chester F. Carlson Center for Imaging Science, Digital Imaging and Remote Sensing (DIRS) Laboratory

The software source code contained in this document is the intellectual property of Rochester Institute of Technology (RIT) and is considered confidential information.

2 Overview

This document describes the `CDGenericPlatform` class which implements a `CDPlatform` object (abstract interface for the platform model) that will be commonly used with the DIRSIG4 model. The platform models an imaging system platform that can move as a function of time. The user describes this

motion by providing a location and orientation history of the platform as a function of time. The primary job of this class is to translate and rotate the line-of-site (LOS) of an instrument (see the `CDInstrument` derived classes) or the instrument mount (see the `CDInstrumentMount` derived classes) that holds an instrument. This generic platform model also supports uncertainty in the location and orientation. This means that the user can supply actual position and orientation data and uncertainty metrics for each parameter, and the model will add noise to the predicted position and orientation values.

The units of the platform location are always in meters and are relative to the scene origin. The platform is assumed to be heading down the +Y axis with the +Z axis defined as up. The orientation angles of the platform are based on absolute, right-handed rotations about the X, Y and Z axes. The axes in DIRSIG4 are defined as +X is East, +Y is North and +Z is upward vertical elevation.

The actual axis rotation transforms are described in the `CDMatrix` class which is the low-level, geometry class used to handle 3D vector matrix algebra. These transforms are included here for clarity:

$$M_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & \sin(\theta_x) & \cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

$$M_y = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

$$M_z = \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

where θ_x , θ_y and θ_z are the right-handed angles of rotation about the X, Y and Z axes, respectively. This platform model is flexible because it allows the user to define the order that the axis rotations are performed in. For example, if the requested rotation order is Z, then Y and then X, then the equivalent matrix-vector operation is:

$$\vec{v}_{\text{los}} = M_x(\theta_x) \cdot M_y(\theta_y) \cdot M_z(\theta_z) \cdot \vec{v}_{\text{los}} \quad (4)$$

$$= [M_x(\theta_x) \cdot [M_y(\theta_y) \cdot [M_z(\theta_z) \cdot \vec{v}_{\text{los}}]]] \quad (5)$$

where \vec{v}_{los} is the original line-of-site vector of the instrument or instrument mount.

The platform location/orientation data is read from an supplied XML file. An example XML file is shown below. The `method` element is reserved for future expansion when the method and parameters used to generate specific platform positioning and pointing data (in the platform editing GUI) will be stored here. For now, the `type` attribute for the `method` expects to see `raw` which is a free-style method that is used to describe any other means used to create the data (including hand generated or imported from other sources).

The optional `uncertainty` element contains a description of the uncertainty associated with each location and orientation parameter. At this time, the model only supports a zero meaned, normal distributed noise about the supplied data. For each parameter, the user can supply the `variance` for that normal distribution. At a future time, a `mean` value may be added that would model an offset in the measured values. The `uncertainty` element has `spatialunits` and `angularunits` attributes that define the units for the variance values and currently only meters and radians are supported, respectively.

The `data` element contains a series of `entry` elements that describe the platform location and orientation at a specific set of times. The `data` element includes the `rotationorder` attribute which is assigned a string that indicates the order that the axis rotations are performed in. For example, the string `zyx` indicates the order is Z, then Y and then X (with the actual matrix multiplication order being the opposite when read left to right). The `spatialunits` attribute is used to define the units of all the location values, and currently only supports meters. The `angularunits` attribute is used to define the units of all the rotation angles, and currently only supports radians.

Within each `entry` element is the time for this waypoint and the location and orientation. The `datetime` element is read by the `CDDateTime::load(QDomElement&)` function which supports both absolute and relative times. The X, Y and Z locations are in scene units. The X, Y and Z rotations are assumed to be absolute and in radians (currently).

```
3  <Example generic platform position data (PPD) file 3>≡
    <platform type="generic">
      <method type="raw">
      </method>

      <uncertainty enabled="1" spatialunits="meters" angularunits="radians">
        <xlocation type="normal">
          <variance>0.1</variance>
        </xlocation>
        <ylocation type="normal">
          <variance>0.1</variance>
        </ylocation>
        <zlocation type="normal">
          <variance>0.2</variance>
        </zlocation>
        <xrotation type="normal">
```

```

        <variance>0.01</variance>
    </xrotation>
    <yrotation type="normal">
        <variance>0.01</variance>
    </yrotation>
    <zrotation type="normal">
        <variance>0.01</variance>
    </zrotation>
</uncertainty>

<data rotationorder="zyx" angletype="absolute" spatialunits="meters" angularunits="degrees">
    <entry>
        <datetime type="relative">1.000</datetime>
        <xlocation>0.0</xlocation>
        <ylocation>0.0</ylocation>
        <zlocation>0.0</zlocation>
        <xrotation>0.0</xrotation>
        <yrotation>0.0</yrotation>
        <zrotation>0.0</zrotation>
    </entry>
    <entry>
        <datetime type="relative">2.000</datetime>
        <xlocation>1.0</xlocation>
        <ylocation>1.0</ylocation>
        <zlocation>1.0</zlocation>
        <xrotation>3.1415</xrotation>
        <yrotation>3.1415</yrotation>
        <zrotation>3.1415</zrotation>
    </entry>
</data>
</platform>

```

2.1 Document Version

The following chunk stores the current document version. This is accessible through the class namespace so that the programmer can set up query mechanisms from the application to track the versions used in a specific build.

```

4  <CDGenericPlatform Document version 4>≡ (33)
    const char * CDGenericPlatform::version = "$Revision: 1.26 $";

```

3 The CDGenericPlatform class

3.1 Parent Class

This class is derived from the CDPlatform base class.

3.2 Class Dependencies

This class uses the following classes as private data members and for interfaces through public member functions:

- The CDPlatformData class is used to store the position, orientation and transform of the platform at a specific point in time.
- The CDPoint and CDVector class are used to store 3D point and vector data, respectively.
- The CDMatrix class is used to store 3D transformations.

3.3 Class Implementation

```

5  <CDGenericPlatform class 5>≡ (32)
    class CDGenericPlatform : public CDPlatform
    {
    public:
        <CDGenericPlatform Public Type Definitions 6a>
        <CDGenericPlatform Constructor Declarations 8c>
        <CDGenericPlatform Destructor Declaration 9b>
        <CDGenericPlatform Public Inline Member Functions 9d>
        <CDGenericPlatform Public Member Function Declarations 12c>

    private:
        <CDGenericPlatform Private Variable Declarations 6b>

    public:
        static const char * version;
    };

```

3.4 Public Type Definitions

3.4.1 The orientation angle reference

For each platform position and orientation record includes the orientation of the platform. These orientation angles can be either relative to an absolute coordinate system or relative to the mean flight line. This platform model supports both absolute orientation

```
6a <CDGenericPlatform Public Type Definitions 6a>≡ (5)
    typedef enum {
        UnknownAngles, AbsoluteAngles, FlightRelativeAngles
    } AngleReferenceType;
```

Defines:

`AbsoluteAngles`, used in chunk 21.
`AngleReferenceType`, used in chunks 6b, 9d, and 10a.
`FlightRelativeAngles`, used in chunk 21.
`UnknownAngles`, used in chunks 9a and 21.

3.5 Private Variable Declarations

3.5.1 The orientation angle reference type

This variable stores the orientation type convention used for the various orientation angles.

```
6b <CDGenericPlatform Private Variable Declarations 6b>≡ (5) 6c▷
    AngleReferenceType angleType;
```

Defines:

`angleType`, used in chunks 9, 10a, and 21.

Uses `AngleReferenceType` 6a.

3.5.2 Rotation order

This platform model allows the user to specify the order of the axis rotation transforms. This is stored as a simple 3 character array that contains strings like "xyz" to indicate that the X, then Y and then Z rotation are to be applied.

```
6c <CDGenericPlatform Private Variable Declarations 6b>+≡ (5) <6b 7a▷
    std::string rotationOrder;
```

Defines:

`rotationOrder`, used in chunks 9a, 13, 19, 21, 29, and 34.

3.5.3 The list of known positions/orientations

This variable stores the known position and orientation records in a list managed by an STL `<vector>` of `CDPlatformData` instances. These position/orientation records include a time stamp so they can be interpolated on-the-fly to arbitrary times.

```
7a <CDGenericPlatform Private Variable Declarations 6b>+≡ (5) <6c 7b>
    std::vector< CDPlatformData > positionVector;
```

Defines:

`positionVector`, used in chunks 9–11, 13, 15, 16, 21, and 29.

3.5.4 The cache of computed positions/orientations

This variable stores the position and orientation records computed for specific date/times. It is managed by an STL `<map>` of `CDPlatformData` instances hashed by `CDDateTime` instances. The cache is used by the main accessor function to avoid computing the platform data for a specific time repeatedly.

```
7b <CDGenericPlatform Private Variable Declarations 6b>+≡ (5) <7a 7c>
    std::map< CDDateTime, CDPlatformData > positionCache;
```

Defines:

`positionCache`, used in chunks 9a and 16.

3.5.5 The uncertainty parameters

These variables store the uncertainty or variance in the platform parameters.

```
7c <CDGenericPlatform Private Variable Declarations 6b>+≡ (5) <7b 8a>
    bool enableUncertainty;
    double xLocationVariance;
    double yLocationVariance;
    double zLocationVariance;
    double xRotationVariance;
    double yRotationVariance;
    double zRotationVariance;
```

Defines:

`xLocationVariance`, used in chunks 9a, 16, and 21.

`xRotationVariance`, used in chunks 9a, 16, 21, and 29.

`yLocationVariance`, used in chunks 9a, 16, and 21.

`yRotationVariance`, used in chunks 9a, 16, 21, and 29.

`zLocationVariance`, used in chunks 9a, 16, and 21.

`zRotationVariance`, used in chunks 9a, 16, 21, and 29.

3.5.6 The average altitude

These variables store the average altitude of the platform.

```
8a  <CDGenericPlatform Private Variable Declarations 6b>+≡ (5) <7c 8b>
      double averageAltitude;
```

Defines:

averageAltitude, used in chunks 9a, 11c, and 13.

3.5.7 The average orientation angles

These variables store the average rotation angles for the three primary dimensions. Like the rotation angles, these angles are also in radians.

```
8b  <CDGenericPlatform Private Variable Declarations 6b>+≡ (5) <8a
      double averageAngleX;
      double averageAngleY;
      double averageAngleZ;
```

Defines:

averageAngleX, used in chunks 9a, 11d, and 13.

averageAngleY, used in chunks 9a, 12a, and 13.

averageAngleZ, used in chunks 9a, 12b, and 13.

3.6 Constructors

3.6.1 Default Constructor

This default constructor is provided so that the user can create an “empty” platform location history.

```
8c  <CDGenericPlatform Constructor Declarations 8c>≡ (5)
      CDGenericPlatform( void );
```

9a \langle CDGenericPlatform Constructor Implementations 9a $\rangle \equiv$ (33)

```

CDGenericPlatform::CDGenericPlatform( void )
    : CDPlatform(), angleType( UnknownAngles ), rotationOrder( "zyx" ),
      positionVector(), positionCache(),
      enableUncertainty( false ),
      xLocationVariance( 0.0 ), yLocationVariance( 0.0 ),
      zLocationVariance( 0.0 ),
      xRotationVariance( 0.0 ), yRotationVariance( 0.0 ),
      zRotationVariance( 0.0 ),
      averageAltitude( 0.0 ),
      averageAngleX( 0.0 ), averageAngleY( 0.0 ), averageAngleZ( 0.0 )
{
}

```

Defines:

CDGenericPlatform::CDGenericPlatform(void), never used.

Uses angleType 6b, averageAltitude 8a, averageAngleX 8b, averageAngleY 8b, averageAngleZ 8b, positionCache 7b, positionVector 7a, rotationOrder 6c, UnknownAngles 6a, xLocationVariance 7c, xRotationVariance 7c, yLocationVariance 7c, yRotationVariance 7c, zLocationVariance 7c, and zRotationVariance 7c.

3.7 Destructor

The destructor for this class is simple since there is no dynamically allocated data in this class. The STL `<vector>` template will take care of freeing the memory associated with the position/orientation data.

9b \langle CDGenericPlatform Destructor Declaration 9b $\rangle \equiv$ (5)

```

~CDGenericPlatform( void );

```

9c \langle CDGenericPlatform Destructor Implementation 9c $\rangle \equiv$ (33)

```

CDGenericPlatform::~CDGenericPlatform( void )
{
}

```

Defines:

CDGenericPlatform::~CDGenericPlatform(void), never used.

3.8 Accessors and Mutators

3.8.1 Set and get the orientation type

9d \langle CDGenericPlatform Public Inline Member Functions 9d $\rangle \equiv$ (5) 10a \triangleright

```

inline void setAngleType( AngleReferenceType _angleType ) {
    this->angleType = _angleType;
}

```

Defines:

CDGenericPlatform::setAngleType(AngleReferenceType), never used.

Uses AngleReferenceType 6a and angleType 6b.

10a \langle CDGenericPlatform Public Inline Member Functions 9d $\rangle + \equiv$ (5) \langle 9d 10b \rangle

```

    inline AngleReferenceType getAngleType( void ) const {
        return this->angleType;
    }

```

Defines:

CDGenericPlatform::setAngleType(AngleReferenceType), never used.

Uses AngleReferenceType 6a and angleType 6b.

3.8.2 Clear the current data entries

This function will clear/delete all the platform data entries currently stored in the internal data storage.

10b \langle CDGenericPlatform Public Inline Member Functions 9d $\rangle + \equiv$ (5) \langle 10a 10c \rangle

```

    inline void clear( void ) {
        this->positionVector.clear();
    }

```

Defines:

CDGenericPlatform::clear(void), never used.

Uses positionVector 7a.

3.8.3 Set and get the uncertainty enable flag

This pair of mutator and accessor functions will manipulate the flag to enable the uncertainty modeling.

10c \langle CDGenericPlatform Public Inline Member Functions 9d $\rangle + \equiv$ (5) \langle 10b 10d \rangle

```

    inline void setEnableUncertainty( bool _enableUncertainty ) {
        this->enableUncertainty = _enableUncertainty;
    }

```

Defines:

CDGenericPlatform::setEnableUncertainty(bool), never used.

10d \langle CDGenericPlatform Public Inline Member Functions 9d $\rangle + \equiv$ (5) \langle 10c 11a \rangle

```

    inline bool getEnableUncertainty( void ) {
        return this->enableUncertainty;
    }

```

Defines:

CDGenericPlatform::getEnableUncertainty(void), never used.

3.8.4 Add a data entry

This simple function adds a position entry (instance of the CDPlatformData class) to the list of entries.

```
11a <CDGenericPlatform Public Inline Member Functions 9d>+≡ (5) <10d 11b>
      inline void add( const CDPlatformData & pos ) {
          this->positionVector.push_back( pos );
      }
```

Defines:

CDGenericPlatform::add(CDPlatformData&), never used.

Uses positionVector 7a.

3.8.5 Get number of position entries

This accessor is outdated and only implemented to provide backwards compatibility for the time being.

```
11b <CDGenericPlatform Public Inline Member Functions 9d>+≡ (5) <11a 11c>
      inline int getCount( void ) const {
          return this->positionVector.size();
      }
```

Defines:

CDGenericPlatform::getCount(void), never used.

Uses positionVector 7a.

3.8.6 Get average orientation angles

This function returns the average altitude of the platform data.

```
11c <CDGenericPlatform Public Inline Member Functions 9d>+≡ (5) <11b 11d>
      inline double getAverageAltitude( void ) const {
          return this->averageAltitude;
      }
```

Defines:

CDGenericPlatform::getAverageAltitude(void), never used.

Uses averageAltitude 8a.

3.8.7 Get average orientation angles

These functions should return the average angles for all of the platform data.

```
11d <CDGenericPlatform Public Inline Member Functions 9d>+≡ (5) <11c 12a>
      inline double getAverageAngleX( void ) const {
          return this->averageAngleX;
      }
```

Defines:

CDGenericPlatform::getAverageAngleX(void), never used.

Uses averageAngleX 8b.

12a \langle CDGenericPlatform Public Inline Member Functions 9d $\rangle + \equiv$ (5) \langle 11d 12b \rangle

```

    inline double getAverageAngleY( void ) const {
        return this->averageAngleY;
    }

```

Defines:

CDGenericPlatform::getAverageAngleY(void), never used.

Uses averageAngleY 8b.

12b \langle CDGenericPlatform Public Inline Member Functions 9d $\rangle + \equiv$ (5) \langle 12a \rangle

```

    inline double getAverageAngleZ( void ) const {
        return this->averageAngleZ;
    }

```

Defines:

CDGenericPlatform::getAverageAngleZ(void), never used.

Uses averageAngleZ 8b.

3.8.8 Initialization function

This function is called to initialize the platform position data for the run-time simulation. In general, it will attempt to open the position and orientation file indicated by the `filename` variable, read in the data, and compute the LOS transform matrix using the define convention.

12c \langle CDGenericPlatform Public Member Function Declarations 12c $\rangle \equiv$ (5) 15a \rangle

```

    bool init( CDSimulation * simulation );

```

```

13  <CDGenericPlatform Public Member Function Implementations 13>≡ (33) 15b>
    bool
    CDGenericPlatform::init( CDSimulation * simulation )
    {
        // talk to the people
        std::cout
            << "Initializing generic platform model: " << std::endl;

        // check if this object has a valid filename
        if( this->getFilename().empty() ) {
            std::cerr
                << " No data filename was supplied for this platform model!"
                << std::endl;
            return false;
        }

        // talk to the people
        std::cout
            << " Location/Orientation history filename = "
            << this->getFilename() << std::endl;

        // read in the data
        if( !this->load( this->getFilename() ) ) {
            std::cerr
                << "Generic Platform: File Read Error!" << std::endl
                << " Could not load position file!" << std::endl;
            return false;
        }

        // make sure the rotation order is correct
        if( this->rotationOrder.length() != 3 ) {
            std::cerr
                << "Generic Platform: File Read Error!" << std::endl
                << " Rotation order attribute is the wrong size!" << std::endl;
            return false;
        }

        // iterate through each entry, convert relative times to absolute
        this->averageAltitude = 0.0;
        for( unsigned int ii = 0; ii < this->positionVector.size(); ii++ ) {

            // compute a default transform for each entry
            this->computeTransform( this->positionVector[ ii ] );

            // check if the time was relative

```

```

    if( this->positionVector[ ii ].getDateTime().getType()
        == CDDateTime::RelativeDateTime ) {

        // calculate the absolute date/time
        CDDateTime absDateTime = this->getStartDateTime() +
            this->positionVector[ ii ].getDateTime().getTotalSeconds();

        // store the new absolute time
        this->positionVector[ ii ].setDateTime( absDateTime );

    }

    // accumulate the average altitude
    this->averageAltitude +=
        this->positionVector[ ii ].getActualLocation().getZ();
}

// normalize the average altitude
this->averageAltitude /= this->positionVector.size();

// get the first and last entry
CDPlatformData firstEntry = this->positionVector[ 0 ];
CDPlatformData lastEntry =
    this->positionVector[ this->positionVector.size() - 1 ];

// compute the overall flight delta values
double deltaX = lastEntry.getActualLocation().getX() -
    firstEntry.getActualLocation().getX();
double deltaY = lastEntry.getActualLocation().getY() -
    firstEntry.getActualLocation().getY();

// compute the flight average values
this->averageAngleX = 0.0;
this->averageAngleY = 0.0;
this->averageAngleZ = -atan2( deltaX, deltaY );

std::cout << "    This file contained " << this->positionVector.size()
    << " location/orientation records" << std::endl << std::endl;

return true;
}

```

Defines:

CDGenericPlatform::init(CDSimulation*), never used.

Uses averageAltitude 8a, averageAngleX 8b, averageAngleY 8b, averageAngleZ 8b,
positionVector 7a, and rotationOrder 6c.

3.8.9 Get the position data at a specific index

15a \langle CDGenericPlatform Public Member Function Declarations 12c \rangle + \equiv (5) \langle 12c 15c \rangle
`const CDPlatformData & index(unsigned int index) const;`

15b \langle CDGenericPlatform Public Member Function Implementations 13 \rangle + \equiv (33) \langle 13 15d \rangle
`const CDPlatformData &
 CDGenericPlatform::index(unsigned int index) const
 {
 return this->positionVector[index];
 }`

Defines:

CDGenericPlatform::index(uint), never used.

Uses positionVector 7a.

15c \langle CDGenericPlatform Public Member Function Declarations 12c \rangle + \equiv (5) \langle 15a 15e \rangle
`CDPlatformData & index(unsigned int index);`

15d \langle CDGenericPlatform Public Member Function Implementations 13 \rangle + \equiv (33) \langle 15b 16 \rangle
`CDPlatformData &
 CDGenericPlatform::index(unsigned int index)
 {
 return this->positionVector[index];
 }`

Defines:

CDGenericPlatform::index(uint), never used.

Uses positionVector 7a.

3.8.10 Get the position for a specific time

This function implements the virtual accessor defined in the CDPlatform base class. It will interpolate an instance of the CDPlatformData (which contains a position, orientation and LOS transform) for the supplied absolute data/time (see the dateTime parameter).

15e \langle CDGenericPlatform Public Member Function Declarations 12c \rangle + \equiv (5) \langle 15c 18 \rangle
`const CDPlatformData get(const CDDateTime & dateTime);`

```

16  <CDGenericPlatform Public Member Function Implementations 13>+≡ (33) <15d 19>
    const CDPlatformData
    CDGenericPlatform::get( const CDDateTime & dateTime )
    {
        // if there is only one point, return that
        if( this->positionVector.size() == 1 ) {
            return this->positionVector[ 0 ];
        }

        // check data for this date/time is in the cache already
        std::map< CDDateTime, CDPlatformData >::iterator cacheIter;
        cacheIter = this->positionCache.find( dateTime );
        if( cacheIter != this->positionCache.end() ) {
            return cacheIter->second;
        }

        // look for the closest point
        unsigned int ii = 0;
        while( ( ii < this->positionVector.size() )
            && ( this->positionVector[ ii ].getDate() < dateTime ) ) {
            ii += 1;
        }

        // if we ran out of entries, back up to the 2nd to last
        if( ii >= this->positionVector.size() ) {
            ii = this->positionVector.size() - 2;
        }
        // if we overshot by time, back up one entry if we can
        else if( ii > 0 ) {
            ii -= 1;
        }

        // get the two times
        CDDateTime time1 = this->positionVector[ ii ].getDate();
        CDDateTime time2 = this->positionVector[ ii+1 ].getDate();

        // compute the weighting factor
        double weight = ( dateTime - time1 ) / ( time2 - time1 );

        // compute the new position
        CDPlatformData currentPosition =
            ( this->positionVector[ ii+1 ] * weight )
            + ( this->positionVector[ ii ] * ( 1.0 - weight ) );

        // compute the full transform matrix
        this->computeTransform( currentPosition );
    }

```

```

// check if we need to compute the "lies" (uncertainty) values
if( this->enableUncertainty ) {
    CDRandomNumber randGen;

    // compute the uncertainty in the location
    CDPoint locationUncertainty;
    locationUncertainty.setX(
        randGen.getGaussian() * this->xLocationVariance );
    locationUncertainty.setY(
        randGen.getGaussian() * this->yLocationVariance );
    locationUncertainty.setZ(
        randGen.getGaussian() * this->zLocationVariance );

    // add this uncertainty to the actual location
    currentPosition.setUncertainLocation(
        currentPosition.getActualLocation() + locationUncertainty );

    // compute the rotation uncertainty
    currentPosition.setUncertainXRotation(
        currentPosition.getActualXRotation()
        + randGen.getGaussian() * this->xRotationVariance );
    currentPosition.setUncertainYRotation(
        currentPosition.getActualYRotation()
        + randGen.getGaussian() * this->yRotationVariance );
    currentPosition.setUncertainZRotation(
        currentPosition.getActualZRotation()
        + randGen.getGaussian() * this->zRotationVariance );
}
else {
    // just set the uncertain values the same as the actual values
    currentPosition.setUncertainLocation(
        currentPosition.getActualLocation() );
    currentPosition.setUncertainXRotation(
        currentPosition.getActualXRotation() );
    currentPosition.setUncertainYRotation(
        currentPosition.getActualYRotation() );
    currentPosition.setUncertainZRotation(
        currentPosition.getActualZRotation() );
}

// add current position/orientation for this date time to the cache
this->positionCache.insert( std::make_pair( dateTime, currentPosition ) );

return currentPosition;
}

```

Defines:

`CDGenericPlatform::get(CDDateTime&)`, never used.

Uses `positionCache 7b`, `positionVector 7a`, `xLocationVariance 7c`, `xRotationVariance 7c`,
`yLocationVariance 7c`, `yRotationVariance 7c`, `zLocationVariance 7c`,
and `zRotationVariance 7c`.

3.8.11 Compute the full transform [private]

This function will compute the full axis rotation matrix using the rotation order (see the `rotationOrder` variable) defined in the overview of this class. This function computes the final/full rotation matrix for the supplied X, Y and Z rotation angles by multiplying individual axis rotation matrices created by the `CDMatrix::makeXRotation(double)`, `CDMatrix::makeYRotation(double)` and `CDMatrix::makeZRotation(double)` functions.

```
18  <CDGenericPlatform Public Member Function Declarations 12c>+≡      (5) <15e 20a>
      void computeTransform( CDPlatformData & entry );
```

```

19  <CDGenericPlatform Public Member Function Implementations 13>+≡ (33) <16 20b>
void
CDGenericPlatform::computeTransform( CDPlatformData & entry )
{
    // create the axis rotation matrices
    CDMatrix xRotateTransform, yRotateTransform, zRotateTransform;
    xRotateTransform.makeXRotation( entry.getActualXRotation() );
    yRotateTransform.makeYRotation( entry.getActualYRotation() );
    zRotateTransform.makeZRotation( entry.getActualZRotation() );

    // start with an identity transform
    CDMatrix finalTransform;

    // apply each axis transform in order
    for( int ii = 0; ii < 3; ii++ ) {
        switch( this->rotationOrder[ ii ] ) {
            case 'x':
                finalTransform = xRotateTransform * finalTransform;
                break;

            case 'y':
                finalTransform = yRotateTransform * finalTransform;
                break;

            case 'z':
                finalTransform = zRotateTransform * finalTransform;
                break;

        }
    }

    // store the final transform into the entry
    entry.setMatrix( finalTransform );

    return;
}

```

Defines:

CDGenericPlatform::computeTransform(CDPlatformData&), never used.

Uses rotationOrder 6c.

3.8.12 Load data from an XML file

For this function, the caller provides the name of a file that contains only the detector response XML description. The file is opened and then the `CDDetectorResponse::load(QDomElement)` function is called to parse the contents of it.

20a `<CDGenericPlatform Public Member Function Declarations 12c>+≡ (5) <18 20c>`
`bool load(const std::string & filename);`

20b `<CDGenericPlatform Public Member Function Implementations 13>+≡ (33) <19 21>`
`bool`
`CDGenericPlatform::load(const std::string & _filename)`
`{`
`QDomDocument doc;`
`QFile file(_filename.c_str());`
`if(!file.open(IO_ReadOnly)) {`
`return false;`
`}`
`if(!doc.setContent(&file)) {`
`file.close();`
`return false;`
`}`
`file.close();`

`// use the document load() function`
`QDomElement docElement = doc.documentElement();`
`return this->load(docElement);`
`}`

Defines:

`CDGenericPlatform::load(string&)`, never used.

3.8.13 Load data from XML document

This function will parse all the detector response information stored in the DOM element passed in.

20c `<CDGenericPlatform Public Member Function Declarations 12c>+≡ (5) <20a 26>`
`bool load(QDomElement & docElement);`

```

21  <CDGenericPlatform Public Member Function Implementations 13>+≡ (33) <20b 27>
    bool
    CDGenericPlatform::load( QDomElement & docElement )
    {
        QDomElement tmpElement;

        // try to extract the "method" section
        QDomElement methodElement = getElement( docElement, "method" );
        if( methodElement.isNull() ) {
            std::cerr
                << "GenericPlatform::load:" << std::endl
                << "    The method section/element is missing!" << std::endl
                << std::endl;
            return false;
        }

        // get method "type" attribute
        QString methodTypeString = methodElement.attribute( "type" );

        // check the method "type" attribute
        if( methodTypeString.isEmpty() ) {
            std::cerr
                << "GenericPlatform::load:" << std::endl
                << "    Missing method type attribute!" << std::endl
                << std::endl;
            return false;
        }
        if( methodTypeString != "raw" ) {
            std::cerr
                << "GenericPlatform::load:" << std::endl
                << "    Unknown method type attribute!" << std::endl
                << "    Type = " << methodTypeString << std::endl
                << std::endl;
            return false;
        }

        // try to extract the optional "uncertainty" section
        QDomElement uncertaintyElement = getSection( docElement, "uncertainty" );
        if( !uncertaintyElement.isNull() ) {

            // get method "enabled" attribute
            QString enableUncertaintyString =
                uncertaintyElement.attribute( "enabled" );

            // set the enableUncertainty

```

```
if( ( enableUncertaintyString.isEmpty() )
|| ( enableUncertaintyString == "0" ) ) {
    this->enableUncertainty = false;
}
else if( enableUncertaintyString == "1" ) {
    this->enableUncertainty = true;
}

// try to pull the xlocation variance
tmpElement = getSection( uncertaintyElement, "xlocation" );
if( !tmpElement.isNull() ) {
    this->xLocationVariance =
        getVariable( tmpElement, "variance" ).toDouble();
}

// try to pull the ylocation variance
tmpElement = getSection( uncertaintyElement, "ylocation" );
if( !tmpElement.isNull() ) {
    this->yLocationVariance =
        getVariable( tmpElement, "variance" ).toDouble();
}

// try to pull the zlocation variance
tmpElement = getSection( uncertaintyElement, "zlocation" );
if( !tmpElement.isNull() ) {
    this->zLocationVariance =
        getVariable( tmpElement, "variance" ).toDouble();
}

// try to pull the xrotation variance
tmpElement = getSection( uncertaintyElement, "xrotation" );
if( !tmpElement.isNull() ) {
    this->xRotationVariance =
        getVariable( tmpElement, "variance" ).toDouble();
}

// try to pull the yrotation variance
tmpElement = getSection( uncertaintyElement, "yrotation" );
if( !tmpElement.isNull() ) {
    this->yRotationVariance =
        getVariable( tmpElement, "variance" ).toDouble();
}

// try to pull the zrotation variance
tmpElement = getSection( uncertaintyElement, "zrotation" );
if( !tmpElement.isNull() ) {
```

```
        this->zRotationVariance =
            getVariable( tmpElement, "variance" ).toDouble();
    }
}

// try to extract the "data" section
QDomElement dataElement = getSection( docElement, "data" );
if( dataElement.isNull() ) {
    std::cerr
        << "GenericPlatform::load:" << std::endl
        << "    The data section/element is missing!" << std::endl
        << std::endl;
    return false;
}

// get "rotationorder" attribute
QString orderString = dataElement.attribute( "rotationorder" );

// check if the order string was empty
if( orderString.isEmpty() ) {
    std::cerr
        << "GenericPlatform::load:" << std::endl
        << "    Missing rotationorder attribute!" << std::endl;
    return false;
}

// check if the order string was not 3 characters long
if( orderString.length() != 3 ) {
    std::cerr
        << "GenericPlatform::load:" << std::endl
        << "    The rotationorder must be exactly 3 characters long"
        << std::endl;
    return false;
}

// store the order string
this->rotationOrder = orderString.ascii();

// get "angletype" attribute
QString typeString = dataElement.attribute( "angletype" );

// check which type this was
if( ( typeString.isEmpty() ) || ( typeString == "absolute" ) ) {
    this->angleType = AbsoluteAngles;
}
```

```
}
else if( typeString == "flightrelative" ) {
    this->angleType = FlightRelativeAngles;
}
else {
    this->angleType = UnknownAngles;
}

// iterate through each entry, calculate the transform
CDPlatformData tmpPosition;
QDomNodeList entryNodeList = dataElement.elementsByTagName( "entry" );
for( unsigned int ii = 0; ii < entryNodeList.count(); ii++ ) {

    // get this entry element
    QDomElement entryElement = entryNodeList.item( ii ).toElement();

    // get and parse the time element
    tmpElement = getElement( entryElement, "datetime" );
    if( tmpElement.isNull() ) {
        std::cerr
            << "GenericPlatform::load:" << std::endl
            << "    Missing datetime element" << std::endl;
        return false;
    }
    tmpPosition.getDateTime().load( tmpElement );

    // get and parse the "xlocation" element
    tmpElement = getElement( entryElement, "xlocation" );
    if( tmpElement.isNull() ) {
        std::cerr
            << "GenericPlatform::load:" << std::endl
            << "    Missing xlocation element" << std::endl;
        return false;
    }
    tmpPosition.getActualLocation().setX( tmpElement.text().toDouble() );

    // get and parse the "ylocation" element
    tmpElement = getElement( entryElement, "ylocation" );
    if( tmpElement.isNull() ) {
        std::cerr
            << "GenericPlatform::load:" << std::endl
            << "    Missing ylocation element" << std::endl;
        return false;
    }
    tmpPosition.getActualLocation().setY( tmpElement.text().toDouble() );
}
```

```

// get and parse the "zlocation" element
tmpElement = getElement( entryElement, "zlocation" );
if( tmpElement.isNull() ) {
    std::cerr
        << "GenericPlatform::load:" << std::endl
        << "    Missing zlocation element" << std::endl;
    return false;
}
tmpPosition.getActualLocation().setZ( tmpElement.text().toDouble() );

// get and parse the "xrotation" element
tmpElement = getElement( entryElement, "xrotation" );
if( tmpElement.isNull() ) {
    std::cerr
        << "GenericPlatform::load:" << std::endl
        << "    Missing xrotation element" << std::endl;
    return false;
}
tmpPosition.setActualXRotation( tmpElement.text().toDouble() );

// get and parse the "yrotation" element
tmpElement = getElement( entryElement, "yrotation" );
if( tmpElement.isNull() ) {
    std::cerr
        << "GenericPlatform::load:" << std::endl
        << "    Missing yrotation element" << std::endl;
    return false;
}
tmpPosition.setActualYRotation( tmpElement.text().toDouble() );

// get and parse the "zrotation" element
tmpElement = getElement( entryElement, "zrotation" );
if( tmpElement.isNull() ) {
    std::cerr
        << "GenericPlatform::load:" << std::endl
        << "    Missing zrotation element" << std::endl;
    return false;
}
tmpPosition.setActualZRotation( tmpElement.text().toDouble() );

// push this temporary entry into the list
this->positionVector.push_back( tmpPosition );
}
return true;
}

```

Defines:

CDGenericPlatform::load(QDomDocument&), never used.
Uses AbsoluteAngles 6a, angleType 6b, FlightRelativeAngles 6a, positionVector 7a,
rotationOrder 6c, UnknownAngles 6a, xLocationVariance 7c, xRotationVariance 7c,
yLocationVariance 7c, yRotationVariance 7c, zLocationVariance 7c,
and zRotationVariance 7c.

3.8.14 Save data to an XML file

26 <CDGenericPlatform Public Member Function Declarations 12c>+≡ (5) <20c 28>
bool store(const std::string & filename);

27 <CDGenericPlatform Public Member Function Implementations 13>+≡ (33) <21 29>

```

bool
CDGenericPlatform::store( const std::string & _filename )
{
    // try and open the output file
    QFile file( _filename.c_str() );
    if( !file.open( IO_WriteOnly | IO_Truncate ) ) {
        return false;
    }

    // store the response to a document
    QDomDocument doc( "dirsig" );

    // create the "platform" element with the "generic" attribute
    QDomElement docElement = doc.createElement( "platform" );
    docElement.setAttribute( "type", "generic" );

    // add this top level element to the doc
    doc.appendChild( docElement );

    // call the element store() function to fill in the guts
    this->store( doc, docElement );

    // stream the document to the output file
    QTextStream fileStream( &file );
    fileStream << doc.toString( 4 );

    // check the status of the file stream
    if( file.status() == IO_WriteError ) {
        return false;
    }

    // close the file
    file.close();

    return true;
}

```

Defines:

```
CDGenericPlatform::store(string&), never used.
```

3.8.15 Save data to an XML document

This function will save the currently stored position and orientation data to the supplied XML DOM document (see the `doc` parameter).

```
28  <CDGenericPlatform Public Member Function Declarations 12c>+≡ (5) <26  
    bool store( QDomDocument & doc, QDomElement & parentElement );
```

```

29  <CDGenericPlatform Public Member Function Implementations 13>+≡ (33) <27
    bool
    CDGenericPlatform::store( QDomDocument & doc, QDomElement & parentElement )
    {
        // create the "method" element with the "type" attribute
        QDomElement methodElement = doc.createElement( "method" );
        methodElement.setAttribute( "type", "raw" );
        parentElement.appendChild( methodElement );

        // create the "uncertainty" element
        QDomElement uncertaintyElement = doc.createElement( "uncertainty" );
        parentElement.appendChild( uncertaintyElement );

        // add the "enabled" attribute
        if( this->enableUncertainty ) {
            uncertaintyElement.setAttribute( "enabled", "1" );
        }
        else {
            uncertaintyElement.setAttribute( "enabled", "0" );
        }

        QDomElement valueElement, varianceElement;
        QDomText tmpText;

        valueElement = doc.createElement( "xlocation" );
        valueElement.setAttribute( "type", "normal" );
        uncertaintyElement.appendChild( valueElement );
        varianceElement = doc.createElement( "variance" );
        tmpText = doc.createTextNode(
            QString::number( this->xRotationVariance, 'f', 10 ) );
        varianceElement.appendChild( tmpText );
        valueElement.appendChild( varianceElement );

        valueElement = doc.createElement( "ylocation" );
        valueElement.setAttribute( "type", "normal" );
        uncertaintyElement.appendChild( valueElement );
        varianceElement = doc.createElement( "variance" );
        tmpText = doc.createTextNode(
            QString::number( this->yRotationVariance, 'f', 10 ) );
        varianceElement.appendChild( tmpText );
        valueElement.appendChild( varianceElement );

        valueElement = doc.createElement( "zlocation" );
        valueElement.setAttribute( "type", "normal" );
        uncertaintyElement.appendChild( valueElement );
        varianceElement = doc.createElement( "variance" );

```

```
tmpText = doc.createTextNode(
    QString::number( this->zRotationVariance, 'f', 10 ) );
varianceElement.appendChild( tmpText );
valueElement.appendChild( varianceElement );

// create the "data" element
QDomElement dataElement = doc.createElement( "data" );
parentElement.appendChild( dataElement );

// add the "rotationorder" attribute
dataElement.setAttribute( "rotationorder", this->rotationOrder.c_str() );

// add the "angletype" attribute
dataElement.setAttribute( "angletype", "absolute" );

// add the "spatialunits" attribute
dataElement.setAttribute( "spatialunits", "meters" );

// add the "angularunits" attribute
dataElement.setAttribute( "angularunits", "radians" );

// iterate through each position entry and store it
QDomElement tmpElement;

for( unsigned int ii = 0; ii < this->positionVector.size(); ii++ ) {

    // get a temporary reference to this position vector
    CDPlatformData & tmpEntry = this->positionVector[ ii ];

    // create an element for this position entry
    QDomElement entryElement = doc.createElement( "entry" );
    dataElement.appendChild( entryElement );

    // create an element for this entry's time
    this->positionVector[ ii ].getDateTime().store( doc, entryElement );

    // create an element for the X, Y and Z positions
    tmpElement = doc.createElement( "xlocation" );
    entryElement.appendChild( tmpElement );
    tmpText = doc.createTextNode(
        QString::number( tmpEntry.getActualLocation().getX(), 'f', 10 ) );
    tmpElement.appendChild( tmpText );

    tmpElement = doc.createElement( "ylocation" );
    entryElement.appendChild( tmpElement );
    tmpText = doc.createTextNode(
```

```

        QString::number( tmpEntry.getActualLocation().getY(), 'f', 10 ) );
tmpElement.appendChild( tmpText );

tmpElement = doc.createElement( "zlocation" );
entryElement.appendChild( tmpElement );
tmpText = doc.createTextNode(
    QString::number( tmpEntry.getActualLocation().getZ(), 'f', 10 ) );
tmpElement.appendChild( tmpText );

// create an element for the X, Y and Z rotations
tmpElement = doc.createElement( "xrotation" );
entryElement.appendChild( tmpElement );
tmpText = doc.createTextNode(
    QString::number( tmpEntry.getActualXRotation(), 'f', 10 ) );
tmpElement.appendChild( tmpText );

tmpElement = doc.createElement( "yrotation" );
entryElement.appendChild( tmpElement );
tmpText = doc.createTextNode(
    QString::number( tmpEntry.getActualYRotation(), 'f', 10 ) );
tmpElement.appendChild( tmpText );

tmpElement = doc.createElement( "zrotation" );
entryElement.appendChild( tmpElement );
tmpText = doc.createTextNode(
    QString::number( tmpEntry.getActualZRotation(), 'f', 10 ) );
tmpElement.appendChild( tmpText );

    }

    return true;
}

```

Defines:

CDGenericPlatform::store(QDomDocument&,QDomElement&), never used.

Uses positionVector 7a, rotationOrder 6c, xRotationVariance 7c, yRotationVariance 7c,
and zRotationVariance 7c.

4 Header file

The following target collects the contents of the `CDGenericPlatform.h` header file.

```
32 <CDGenericPlatform.h 32>≡
    //
    // DO NOT EDIT THIS FILE!
    // This file was generated by noweb(1) from the following file:
    // $Id: CDGenericPlatform.nw,v 1.26 2006/02/27 19:06:14 sdbpci Exp $
    //
    #ifndef _CDGENERICPLATFORM_H_
    #define _CDGENERICPLATFORM_H_
    #include <iostream>
    #include <vector>
    #include <map>
    #include <cmath>
    #include <cassert>
    #include "CDPlatform.h"
    #include "CDDateTime.h"
    #include "CDPoint.h"
    #include "CDVector.h"
    #include "CDXmlFile.h"
    #include "CDPlatformData.h"

    class CDSimulation;
    class QDomDocument;
    class QDomElement;

    <CDGenericPlatform class 5>

    #endif // _CDGENERICPLATFORM_H_
```

5 Source file

The following target collects the contents of the `CDGenericPlatform.cpp` source file.

```
33 <CDGenericPlatform.cpp 33>≡
    //
    // DO NOT EDIT THIS FILE!
    // This file was generated by noweb(1) from the following file:
    // $Id: CDGenericPlatform.nw,v 1.26 2006/02/27 19:06:14 sdbpci Exp $
    //
    #include <iostream>
    #include "CDGenericPlatform.h"
    #include "CDMatrix.h"
    #include "CDSimulation.h"
    #include "CDPathList.h"
    #include "CDGeoBase.h"
    #include "FDPrfFile.h"
    #include "CDTaskManager.h"
    #include "CDRandomNumber.h"

    #include "qfile.h"
    #include "qdom.h"
    #include "qtextstream.h"

    <CDGenericPlatform Document version 4>
    <CDGenericPlatform Constructor Implementations 9a>
    <CDGenericPlatform Destructor Implementation 9c>
    <CDGenericPlatform Public Member Function Implementations 13>
```

6 Revision History

34 *<Revision History 34>*≡

Revision 1.26 2006/02/27 19:06:14 sdbpci
Added "enabled" attribute to the XML sample.

Revision 1.25 2006/02/20 18:56:56 sdbpci
Added the code to the XML store() function to write out the "uncertainty" information. The code for the load() function was already there.

Revision 1.24 2006/01/19 14:54:10 sdbpci
Added the code to read the "uncertainty" section which describes the uncertainty or noise in the reported location/orientation of the platform.

Revision 1.23 2006/01/18 12:27:24 sdbpci
Fixed a bug in the store() function which resulting in the <entry> elements not being children of the <data> element.

Revision 1.22 2006/01/17 13:58:58 sdbpci
Removed the "startTime" variable which was relocated to the CDPlatform parent class.
Updated all the usages of the old private startTime variable to use the CDPlatform startDateTime mutator/accessor functions.
Removed the "filename" variable which was relocated to the CDPlatform parent class.
Added the new variance variables to store the expected random variance (or uncertainty) in any of the location and orientation values. However, these random offsets are not being computed yet.
Added the mutator/accessor functions for the new variance parameters.
Updated the XML load() function to match the informally standardized interfaces to all our factory derived classes. This model is now identified by <platform type="generic"> syntax in the XML. We might want to make an updater program that migrates old generic platform files (.ppd) into the new format.
Added the parsing of the uncertainty/variance values to the XML load() function.
Cleaned up the XML parsing to use the new XML helper functions in CDXmlFile.

Revision 1.21 2006/01/09 20:03:52 sdbpci
Updated to reflect the swap from CDPlatformPosition to CDPlatformData.
Added documentation for how this model will implement uncertainty in the platform location and orientation.

Revision 1.20 2005/06/24 19:04:16 sdbpci
Fixed the section levels for a bunch of documentation.

- Revision 1.19 2005/06/22 02:10:01 dirs
Fixed a broken `std::string` to `QString` conversion (appeared on the Linux platform).
- Revision 1.18 2005/05/27 01:05:06 sdbpci
Added an implementation of the `CDPlatform::getAverageAltitude()` virtual function.
- Revision 1.17 2005/05/16 04:09:36 sdbpci
Major documentation update to reflect the user-defined rotation order feature and to include the matrix rotation formulas from `CDMatrix` in here too.
Added an STL `map<>` to cache platform position data as a function of time. This should save a significant amount of calculations in the `get()` and `computeTransform()` functions when modeling all systems.
- Revision 1.16 2005/05/13 02:18:25 sdbpci
Added a non-const version of the `index()` function.
- Revision 1.15 2005/05/12 15:10:32 sdbpci
Updated the `XML store()` function to use more precision when saving the position and orientation data.
- Revision 1.14 2005/05/10 02:08:14 sdbpci
Added a default `rotationOrder` string in the constructor.
Added a `clear()` function to purge all the platform data currently stored.
Added an `add()` function to add a new `CDPlatformData` entry to the list.
Fixed a bug in the `store()` main loop that wasn't checkin the index counter.
- Revision 1.13 2005/05/10 01:04:58 sdbpci
Fixed the `::getCount()` function to return the actual number of data entries in the internal storage.
Added a `::index()` function that fetched the position data at a given index.
- Revision 1.12 2005/04/19 17:41:01 sdbpci
Added the "rotationorder" attribute to the example in the introduction.
- Revision 1.11 2005/04/16 02:11:38 sdbpci
Updated this class to have a user supplied rotation order which will allow users to specify the order of the axis rotations that they would like to use (the key to this flexible or "generic" platform object")
- Revision 1.10 2005/02/23 21:41:55 dirs
Updated conversion from a `std::string` to a `QString`

Revision 1.9 2005/01/14 13:57:20 sdbpci

The average Z angle is now computed as the negative to match the axis rotation convention of this class.

Revision 1.8 2005/01/13 01:18:53 sdbpci

An update to this class that reflects that the CDPlatformData class changed the internal storage to be axis rotations rather than roll, pitch and yaw.

Revision 1.7 2005/01/04 21:06:59 sdbpci

Moved the matrix transform computation to an inlined, private function. Updated the documation a bit.

Revision 1.6 2005/01/04 20:20:26 sdbpci

Additional fixes to the main ::get() function temporal interpolation.

Revision 1.5 2004/12/07 04:36:07 sdbpci

Added code in the ::init() function to convert relative position times to absolute so interpolations work correctly.

Added the code to the ::get() function so that the correct transform matrix is computed. A similar chunk of code will most likely be needed in the ClassicPlatform model (but uses different rotation conventions and orders than this class).

Revision 1.4 2004/12/07 02:54:29 sdbpci

Updated the interpolation code in the ::get() function to fix the same issues found in CDClassicPlatform::get() function.

Revision 1.3 2004/11/22 23:13:13 sdbpci

Added some more output messages during the init() function.

Revision 1.2 2004/11/18 03:32:45 sdbpci

Added the XML load/store functions and more documentation.

Revision 1.1 2004/11/10 20:12:33 sdbpci

Added the new "generic" platform model which will be the default platform model in DIRSIG4. This model uses absolute, positive, right-handed rotations about the X, Y and Z axis rather than flight line relative roll, pitch and yaw angles.

Uses rotationOrder 6c.