JFT-145-1-DL.SRM-V1

30-September-94

SOFTWARE REFERENCE MANUAL

FOR THE

DIS LIBRARY (DL) CSCI

OF THE

ADA DISTRIBUTED INTERACTIVE SIMULATION (ADIS) SUPPORT SYSTEM

CONTRACT NO. N00421-92-D-0028

CDRL SEQUENCE NO: A009

Prepared for:

Naval Air Warfare Center, Aircraft Division (NAWCAD)
Systems Engineering Test Directorate (SETD)
Manned Flight Simulator (MFS)

Prepared by:

J. F. Taylor, Inc.
Rt. 235 and Maple Rd.
Lexington Park, MD 20653

Authenticated by:	Approved by:
(Contracting Agency)	(Contractor)
(Date)	(Date)

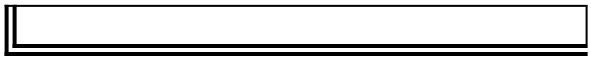

Table of Contents

Tables

Chapter 1: Introduction to the DIS Library (DL)tc "Chapter 1: Introduction to the DIS Library (DL)" \l 1§

What is the DL?tc "What is the DL?" \l 2§

The DIS Library (DL) consists of a set of data processing DIS-related routines that have a broad range of functionallity such as the capability to prioritize or refine a list or stream of Protocal Data Units (a.k.a. PDUs, which represent sets of related simulation data describing an entity or event) that are received from the DIS network, calculate dead-reckoning and smooth entity update positions, and coordinate transformations.

How does the DL work?tc "How does the DL work?"\l 2§

Each routine in the library is independent and has a specific purpose. The units may be used in any desired combination; however, the resulting data reduction and associated processing time may vary with different usages.

What do you need to know to use the DL?tc "What do you need to know to use the DL?"\1 2§

The following prerequisites apply for use of the units within the DL:

- Understanding of how to "with in" or import Ada procedures/packages
- Understanding of DIS coordinate systems, PDU definitions, etc.

What are the special features of the DL?tc "What are the special features of the DL?"\l 2§

A generic double linked list and units to manipulate this list are provided. These generic units form the foundation for other generics to filter and sort PDUs based on a specific criteria. Currently, five PDU types are supported, but the instantiations of the generics can be expanded to include all the PDU types.

Chapter 2: How do you use the DL?tc "Chapter 2: How do you use the DL?"\l 1§

How do you interface with the DL?tc "How do you interface with the DL?"\l 2§

The DL units are designed to be called by the DIS Gateway CSCI or a DIS client. The only interface with the DL is that the units are visible (compiled into the Adalib).

What are the callable routines in the DL?tc "What are the Callable routines in the DL?"\l 2§

This section is divided into the following sub-sections:

- Package/Unit Summary
  Each non-generic package is listed along with its purpose. Each callable unit in the package is then listed along with a brief description of the unit's function.

- Generic Package/Unit Summary
  Each generic package is listed along with its purpose. Each callable unit in the package is then listed along with a brief description of the unit's function.

- Detailed Unit Descriptions
  Each callable unit in the DL is listed by package and unit name in alphabetical order. This section includes the procedure syntax, purpose, input/output descriptions, and a code example.

- Detailed Generic Package/Unit Descriptions
  Each generic package in the DL is listed by package name in alphabetical order. This section includes the data that is needed to instantiate the generic and an example of how the generic can be instantiated and called. The syntax for each callable unit in the generic package is included along with its purpose and the exceptions/status codes that can be raised/returned.

Package/Unit Summary tc "Package/Unit Summary"\l 3§

This section lists all the callable DIS Library packages and units within the packages in alphabetical order. A brief description of the purpose of the package and of each callable unit within the package is included.

Calculate Package
Provides units to calculate azimuth, elevation, distance, and velocity.

Az_And_El
Calculates both the azimuth and elevation of an entity with respect to a reference entity.

Azimuth
Calculates the azimuth of an entity with respect to a reference entity.

Calculate_Azimuth
Calculates the azimuth of a position vector.

Calculate_Elevation
Calculates the elevation of a position vector.

Distance
Calculates the distance between two vector positions.

Elevation
Calculates the elevation of an entity with respect to a reference entity.

Velocity
Calculates the magnitude of a velocity vector.

Coordinate_Conversion Package
Provides units to convert between various coordinate systems such as geocentric, geodetic, entity, and local.

Entity_To_Geocentric_Conversion
Converts from entity to geocentric coordinates.

Entity_To_Geocentric_Vel_Conversion
Converts from entity to geocentric coordinates.

Geocentric_To_Entity_Conversion
Converts from geocentric to entity coordinates.

Geocentric_To_Entity_Vel_Conversion
Converts from geocentric to entity coordinates.

Geocentric_To_Geodetic_Conversion
Converts from geocentric x, y, z coordinates to geodetic latitude, longitude, and altitude.

Geocentric_To_Local_Conversion
Converts from geocentric to local coordinates.

Geodetic_To_Geocentric_Conversion
Converts from geodetic latitude, longitude, and altitude to geocentric x, y, z coordinates.

Geodetic_To_Local_Conversion
Converts from geodetic latitude, longitude, and altitude to local x, y, z coordinates.

Local_To_Geocentric_Conversion
Converts from local to geocentric coordinates.

Local_To_Geodetic_Conversion
Converts from local x, y, z coordinates to geodetic latitude, longitude, and altitude.

Dead-Reckoning Package

Provides a unit to dead-reckon an entity according to the following DIS dead-reckoning models: FPW, FVW, RPW, RVW, and Static (i.e., any model not listed is dead-reckoned with the Static model).

Update_Position
Calculates the entity's next dead-reckoned position.

Filter Package

Provides units which detemine if a PDU's data is within a predefined threshold.

Az_And_El
Determines whether the azimuth and elevation of an entity with respect to a reference entity is within the specified threshold ranges.

Azimuth
Determines whether the azimuth of an entity with respect to a reference entity is within the specified threshold range.

Distance

Determines whether the distance between two entities is within a specified maximum distance threshold value.

Elevation

Determines whether the elevation of an entity with respect to a reference entity is within the specified threshold range.

Maximum_Velocity

Determines whether the velocity magnitude of an entity is within the specified maximum velocity threshold.

Minimum_Velocity

Determines whether the velocity magnitude of an entity is within the specified minimum velocity threshold.

Filter_By_PDU_Component Package

Provides a unit which detemines if a PDU's data is within a predefined threshold.

Az_And_El

Determines whether the azimuth and elevation of an entity with respect to a reference entity is within the specified threshold ranges.

Filter_List Package

Instantiates generic filter packages which filter a list of PDUs based on some specified criteria and threshold. Units that do not meet the filter criteria are deleted from the list.

Entity_State_Az_And_El.Filter_Az_And_El *

Filters a list of Entity State PDUs by azimuth and elevation.

Detonation_Azimuth.Filter_Orientation

Filters a list of Detonation PDUs by azimuth.

Entity_State_Azimuth.Filter_Orientation

Filters a list of Entity State PDUs by azimuth.

Fire_Azimuth.Filter_Orientation

Filters a list of Fire PDUs by azimuth.

Detonation_Distance.Filter_Distance

Filters a list of Detonation PDUs by distance.

Entity_State_Distance.Filter_Distance

Filters a list of Entity State PDUs by distance.

Fire_Distance.Filter_Distance

Filters a list of Fire PDUs by distance.

Laser_Distance.Filter_Distance
Filters a list of Laser PDUs by distance.

Transmitter_Distance.Filter_Distance
Filters a list of Transmitter PDUs by distance.

Detonation_Elevation.Filter_Orientation
Filters a list of Detonation PDUs by elevation.

Entity_State_Elevation.Filter_Orientation
Filters a list of Entity State PDUs by elevation.

Fire_Elevation.Filter_Orientation
Filters a list of Fire PDUs by elevation.

Entity_State_Max_Velocity.Filter_Velocity
Filters a list of Entitiy State PDUs by maximum velocity.

Entity_State_Min_Velocity.Filter_Velocity
Filters a list of Entitiy State PDUs by minimum velocity.

** Unit is overloaded (i.e., same name different parameters/implementation)*

Hashing Package
Contains units to maintain the hash table used by the Smooth_Position_Update package.

Delete_Item
Deletes the entity information from the hash table.

Orientation_Conversions Package
Contains units to convert between Euler Angles and local roll, pitch, and heading and vice versa.

Eulers_To_Local_Orientation
Converts from Euler Angles to local roll, pitch, and heading.

Local_To_Eulers_Orientation
Converts from local roll, pitch and heading to Euler Angles.

Smooth_Position_Update Package
Contains a unit which dead-reckons and smooths entities.

Smooth_Entity
Dead-reckons and smooths the entity position data.

Sort_List Package
Instantiates generic sort packages which sort a list of PDUs based on some specified criteria.

Detonation_Distance.Sort_Distance
Sorts a list of Detonation PDUs by distance.

Entity_State_Distance.Sort_Distance
Sorts a list of Entity State PDUs by distance.

Fire_Distance.Sort_Distance
Sorts a list of Fire PDUs by distance.

Laser_Distance.Sort_Distance
Sorts a list of Laser PDUs by distance.

Transmitter_Distance.Sort_Distance
Sorts a list of Transmitter PDUs by distance.

Entity_State_Velocity.Sort_Velocity
Sorts a list of Entity State PDUs by velocity.

Generic Package/Unit Summary tc "Generic Package/Unit Summary"\l 3§

This section includes all the DIS Library generic packages and units within the packages in alphabetical order. A brief description of the purpose of the package and each callable unit within the package is included.

Generic_Binary_Insertion_Sort Package
Provides a generic package which can be instantiated for any data type and any sort criteria.

Sort
Sorts an array of items by inserting the current item to sort at the proper place within a list of previously sorted items.

Generic_Filter_List_By_Az_And_El Package
Provides a generic package to evaluate each PDU in a list and remove those PDUs which do not meet the specified azimuth and elevation thresholds. This unit can be instantiated for any PDU type.

Filter_Az_And_El*
Filters a list of PDUs by azimuth and elevation.

** Unit is overloaded (i.e., same name different parameters/implementation)*

Generic_Filter_List_By_Distance Package
Generic package to evaluate each PDU in a list and remove those PDUs which do not meet the specified distance threshold. This unit can be instantiated for any PDU type.

Filter_Distance
Filters a list of PDUs by distance.

Generic_Filter_List_By_Orientation Package
Generic package to evaluate each PDU in a list and remove those PDUs which do not meet the specified azimuth or elevation threshold. This unit can be instantiated for any PDU type and either azimuth or elevation orientation.

Filter_Orientation
Filters a list of PDUs by Orientation.

Generic_Filter_List_By_Velocity Package

Generic package to evaluate each PDU in a list and remove those PDUs which do not meet the specified minimum or maximum velocity threshold. This unit can be instantiated for any PDU type and either minimum or maximum velocity.

Filter_Velocity

Filters a list of PDUs by Velocity.

Generic_List Package

Generic package to create and manage a generic double linked, unbounded list of zero or more items in which items can be added to and removed from any position such that a strict linear ordering is maintained.

Change_Item

Clears the current item and assigns the input item.

Clear_List

Sets the input pointer to null.

Clear_Previous

Sets a node's "previous" pointer to null.

Clear_Next

Sets a node's "next" pointer to null.

Clear_Node

Sets both "previous" and "next" pointers of a node to null.

Construct_Bottom

Adds a new node at the end of the list.

Construct_Top

Adds a new node at the head of the list.

Copy

Copies the items from one list to another list.

Free

Deallocates storage and sets the pointer to null.

Set_Head

Assigns the input item.

Swap_Tail
Exchanges the tail of one list for the tail of another list.

Is_Equal
Returns TRUE if the lists have the same state (i.e., equal number of items and the order and value of their items are the same).

Is_Null
Returns TRUE if there are no items in the list.

Length_Of
Returns the number of items in the list (traverses from input point to bottom).

Predecessor_Of
Returns a list containing the sequence of nodes preceding the node pointed to by the input pointer.

Tail_Of
Returns a list containing the sequence of nodes after the node pointed to by the input pointer.

Value_Of
Returns the value of the item at the head of the list.

Generic_List_Utilities Package
Generic package that is built on the more primative Generic_List units which it imports.

Append_List
Concatenates two lists. Appends List_2 to List_1 and sets List_2 to null.

Assign_Item
Assigns the input item to the to the ITEM component of the node pointed to by the input pointer.

Change_The_Item
Clears the current item and assigns the input item.

Check_At_End
Returns TRUE if the input pointer is pointing to the last node in the list.

Check_At_Head
Returns TRUE if the input pointer is pointing to the first node in the list.

Check_List_Equal
Returns TRUE if the lists have the same state (i.e., equal number of items and the order and value of their items are the same).

Check_Null
Null_Pointer is set TRUE if there are no items in the list.

Clear_Previous_Ptr
Sets the node's previous pointer to null.

Clear_Next_Ptr
Sets the node's next pointer to null.

Clear_The_List
Sets the input pointer to null.

Clear_The_Node
Sets both "previous" and "next" pointers of a node to null.

Copy_List
Copy the items from one list to another list.

Delete_Item
Removes an item from a list and returns the pointer to the deleted item.

Delete_Item_And_Free_Storage
Removes an item from a list and deletes the item's storage.

Find_Position_Of_Item
Position is set to the numeric position of the item within the list.

Free_Node
Deallocates storage and sets the pointer to null.

Free_List
Deletes each item from the list and clears the storage for that node. When all the nodes are deleted the list is set to null.

Get_First_Item
Returns a pointer to the item at the head of the list.

Get_Last_Item
Returns a pointer to the last item in the list.

Get_Item
Returns the value of a node's ITEM.

Get_Previous
Returns a list containing the sequence of nodes preceding the node pointed to by the input pointer.

Get_Next
Returns a list containing the sequence of nodes after the node pointed to by the input pointer.

Get_Sublist
Returns a sublist whose head is the item given.

Get_Sublist
Returns a sublist whose head is the item at the input position.

Get_Size
Returns the number of items in the list (i.e., traverses from a point input to bottom).

Insert_Item
Inserts an item into a list after a given position.

Insert_Item_End
Adds an item to the end of the list.

Insert_Item_Top
Adds an item to the top of the list.

Insert_List
Inserts a list in a list after the given position.

Split
Breaks a list into two lists at a given position.

Straight_Insertion_Sort
Examines one item at a time and inserts it in the proper order relative to all previously processed items. Equal key values maintain their order.

Swap_Tails
Exchanges the tail of one list for the tail of another list.

Generic_Sort_List_By_Distance Package
Generic package which instantiates the Generic_Binary_Insertion_Sort for distance. This package can be instantiated for any PDU type.

Sort_Distance
Sorts a list of PDUs by distance.

Generic_Sort_List_By_Velocity Package
Generic package which instantiates the Generic_Binary_Insertion_Sort for velocity. This package can be instantiated for any PDU type.

Sort_Velocity
Sorts a list of PDUs by velocity.

Detailed Unit Descriptions:tc "Detailed Unit Descriptions"\l 3§

This section contains the detailed descriptions of each callable routine in the DIS Library including the instantiations of generic packages. Each callable unit is listed by package and unit name in alphabetical order. This section includes the procedure syntax, purpose, input/output descriptions, and a code example.

Calculate.Az_And_El Unit tc "Calculate.Az_And_El Unit"\l 4§

Procedure Syntax:

```
Calculate.Az_And_El(
  Entity_State_PDU     : in     DIS_Types.AN_ENTITY_STATE_PDU;
  Position_Of_Interest : in     DIS_Types.A_WORLD_COORDINATE;
  Azimuth              :    out Numeric_Types.FLOAT_32_BIT;
  Elevation            :    out Numeric_Types.FLOAT_32_BIT;
  Status               :    out DL_Status.STATUS_TYPE);
```

Remarks:

Calculates the azimuth and elevation of an entity (Position_Of_Interest) with respect to an entity ((Entity_State_PDU). Converts the Position_Of_Interest to the Entity_State_PDU's coordinate system then calculates the azimuth and elevation. The results are an azimuth angle between 0.0 and 360.0 -2^{-23} degrees and an elevation angle between -180.0 $+2^{-23}$ and 180.0 degrees.

Input Parameters:

Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Position_Of_Interest - Holds a geocentric position in meters which is translated into the entity coordinate system of the entity described by the Entity_State_PDU.

Output Parameters:

Azimuth - Represents the relative angle in degrees of the position of interest to the heading of the entity described by the Entity_State_PDU.

Elevation - Represents the relative angle of the position of interest in degrees to the pitch of the entity described by the Entity_State_PDU.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.

- DL_Status.CALC_AZ_AND_EL_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Calculate,
    DL_Status,
    DIS_Types,
    Global_Data,
    Numeric_Types;

procedure Check_Az_And_El is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

  Entitiy_Location  : DIS_Types.A_WORLD_COORDINATE
                                  := (X => 2_000.0,
                                      Y => 4_025.0,
                                      Z => 1_025.0);

(Assume that a reference Entity State PDU with the orientation and location fields initialized is
known);

  Reference_PDU     : DIS_Types.AN_ENTITY_STATE_PDU
                                      := Global_Data.Entity_State_PDU;

  The_Azimuth       : Numeric_Types.FLOAT_32_BIT;
  The_Elevation     : Numeric_Types.FLOAT_32_BIT;

  .
  .

begin

  .
  .
  -- Get the azimuth and elevation
  Calculate.Az_And_El(
    Entity_State_PDU     => Reference_PDU,
    Position_Of_Interest => Entity_Location,
    Azimuth              => The_Azimuth,
    Elevation            => The_Elevation,
    Status               => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Check_Az_And_El;
```

Calculate.Azimuth Unit tc "Calculate.Azimuth Unit"\l 4§

Procedure Syntax:

```
Calculate.Azimuth(
  Entity_State_PDU     : in     DIS_Types.AN_ENTITY_STATE_PDU;
  Position_Of_Interest : in     DIS_Types.A_WORLD_COORDINATE;
  Azimuth              :    out Numeric_Types.FLOAT_32_BIT;
  Status               :    out DL_Status.STATUS_TYPE);
```

Remarks:

Calculates the azimuth of an entity (Position_Of_Interest) with respect to an entity ((Entity_State_PDU). Converts the Position_Of_Interest to the Entity_State_PDU's coordinate system then calculates the azimuth. The result is an angle between 0.0 and 360.0 -2^{-23} degrees.

Input Parameters:

Entity_State_PDU - Describes the reference entity and its geocentric coordinates in meters.

Position_Of_Interest - Holds a geocentric position in meters which is translated into the entity coordinate system of the entity described by the Entity_State_PDU.

Output Parameters:

Azimuth - Represents the relative angle in degrees of the position of interest to the heading of the entity described by the Entity_State_PDU.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.CALC_AZIMUTH_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Calculate,
    DL_Status,
    DIS_Types,
    Global_Data,
    Numeric_Types;

procedure Check_Azimuth is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  Entity_Location : DIS_Types.A_WORLD_COORDINATE
                                  := (X => 2_000.0,
                                      Y => 4_025.0,
                                      Z => 1_025.0);
```

(Assume that a reference Entity State PDU with the orientation and location fields initialized is known);

```
  Reference_PDU   :  DIS_Types.AN_ENTITY_STATE_PDU
                                      := Global_Data.Entity_State_PDU;

  The_Azimuth     : Numeric_Types.FLOAT_32_BIT;

  .
  .

begin

  .
  .
  -- Get azimuth
  Calculate.Azimuth(
    Entity_State_PDU     => Reference_PDU,
    Position_Of_Interest => Entity_Location,
    Azimuth              => The_Azimuth,
    Status               => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Check_Azimuth;
```

Calculate.Calculate_Azimuth Unit tc "Calculate.Calculate_Azimuth Unit"\l 4§

Procedure Syntax:

```
Calculate.Calculate_Azimuth(
  Vector_Position : in     DIS_Types.AN_ENTITY_COORDINATE_VECTOR;
  Azimuth         :    out Numeric_Types.FLOAT_32_BIT;
  Status          :    out DL_Status.STATUS_TYPE);
```

Remarks:

Calculates the azimuth of a position vector using the following formula:

Azimuth = Arctan(y/x)

where:

- Azimuth is the angle in degrees.
- y is the y component of the input position vector.
- x is the x component of the input position vector.

and the following adjustments to bearing are made:

If x is negative, then add 180.0 degrees.
If x is positive and y is negative, then add 360.0 degrees.

The result will be an angle measuring between 0.0 and 360 - 2^{-23} degrees.

Input Parameters:

Vector_Position - An entity coordinate vector position in meters.

Output Parameters:

Azimuth - Angle calculated for the input position in degrees.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.CALCULATE_AZ_FAILURE - Indicates an exception was raised in this unit.

Example:

```
with Calculate,
    DL_Status,
    DIS_Types,
    Numeric_Types;

procedure Check_Azimuth is

  -- Declare local variables

  Call_Status : DL_Status.STATUS_TYPES;

  Location    : DIS_Types.AN_ENTITY_COORDINATE_VECTOR
                              :=(X => 200.0,
                                 Y => 425.0,
                                 Z => 125.0);

  The_Azimuth : Numeric_Types.FLOAT_32_BIT;

  .
  .

begin

  .
  .
  --Get azimuth.
  Calculate.Calculate_Azimuth(
    Vector_Position => Location,
    Azimuth         => The_Azimuth,
    Status          => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
  .
    end Check_Azimuth;
```

Calculate.Calculate_Elevation Unit tc "Calculate.Calculate_Elevation Unit"\l 4§

Procedure Syntax:

```
Calculate.Calculate_Elevation(
  Vector_Position : in     DIS_Types.AN_ENTITY_COORDINATE_VECTOR;
  Elevation       :    out Numeric_Types.FLOAT_32_BIT;
  Status          :    out DL_Status.STATUS_TYPE);
```

Remarks:

Calculates the elevation of a position vector using the following formula:

Elevation = Arctan (z/x)

where:

- Elevation is an angle in degrees
- z is the z component of the input position.
- x is the x component of the input position.

and the following adjustments to bearing are made:

If x is positive, then reverse the sign of elevation.
If x is negative and z is positive, then subtract 90.0 degrees.
If x is negative and z is negative, then add 90.0 degrees.

The result will be an angle measuring between $-180.0 + 2^{-23}$ and +180.0 degrees.

Input Parameters:

Vector_Position - An entity coordinate vector position in meters.

Output Parameters:

Elevation - Angle calculated for the input position in degrees.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.CALCULATE_EL_FAILURE - Indicates an exception was raised in this unit.

Example:

```
with Calculate,
    DL_Status,
    DIS_Types,
    Numeric_Types;

procedure Check_Elevation is

  -- Declare local variables

  Call_Status   : DL_Status.STATUS_TYPES;

  Location      : DIS_Types.AN_ENTITY_COORDINATE_VECTOR
                                  := (X => 2_000.0,
                                      Y => 4_025.0,
                                      Z => 1_025.0);

  The_Elevation : Numeric_Types.FLOAT_32_BIT;

  .
  .

begin

  .
  .
  --Get elevation
  Calculate.Calculate_Elevation(
    Vector_Position => Location,
    Elevation       => The_Elevation,
    Status          => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
  .
    end Check_Elevation;
```

Calculate.Distance Unit tc "Calculate.Distance Unit"\l 4§

Procedure Syntax:

```
Calculate.Distance(
  Position_1 : in     DIS_Types.A_WORLD_COORDINATE;
  Position_2 : in     DIS_Types.A_WORLD_COORDINATE;
  Distance   :    out Numeric_Types.FLOAT_64_BIT;
  Status     :    out DL_Status.STATUS_TYPE);
```

Remarks:

Calculates the distance between two vector positions, each of which represents entities in the same orthogonal coordinate system, using the following formula:

µ §

where:

- *d* is the distance between the two positions.

µSYMBOL 183 \f "Symbol" \s 10 \h§ are the x, y, z components of the first position.

§

µSYMBOL 183 \f "Symbol" \s 10 \h§ are the x, y, z components of a second position.

§

Input Parameters:

Position_1 - An entity's vector position in meters.

Position_2 - An entity's vector position in meters.

Output Parameters:

Distance - The distance in meters between the two input positions.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.

- DL_Status.CALC_DISTANCE_FAILURE - Indicates an exception was raised in this unit.

Example:

```
with Calculate,
    DL_Status,
    DIS_Types,
    Numeric_Types;

procedure Check_Distance is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  First_Position  : DIS_Types.A_WORLD_COORDINATE
                                  := (X => 200.0,
                                      Y => 425.0,
                                      Z => 125.0);

  Second_Position : DIS_Types.A_WORLD_COORDINATE
                                  := (X => 250.0,
                                      Y => 450.0,
                                      Z => 130.0);

  The_Distance    : Numeric_Types.FLOAT_64_BIT;

  .
  .

begin

  .
  .
  -- Get the distance between the first and second positions.
  Calculate.Distance(
    Position_1 => First_Position,
    Position_2 => Second_Position,
    Distance   => The_Distance,
    Status     => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Check_Distance;
```

Calculate.Elevation Unit tc "Calculate.Elevation Unit"\l 4§

Procedure Syntax:

```
Calculate.Elevation(
  Entity_State_PDU     : in     DIS_Types.AN_ENTITY_STATE_PDU;
  Position_Of_Interest : in     DIS_Types.A_WORLD_COORDINATE;
  Elevation            :    out Numeric_Types.FLOAT_32_BIT;
  Status               :    out DL_Status.STATUS_TYPE);
```

Remarks:

Calculates the elevation of an entity (Position_Of_Interest) with respect to an entity (Entity_State_PDU) . Converts the Position_Of_Interest to the Entity_State_PDU's coordinate system then calculates the elevation. The result is angle between -180.0 $+2^{-23}$ and 180.0 degrees.

Input Parameters:

Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Position_Of_Interest - Holds a geocentric position in meters which is translated into the entity coordinate system of the entity described by the Entity_State_PDU.

Output Parameters:

Elevation - Represents the relative angle of the position of interest in degrees to the pitch of the entity described by the Entity_State_PDU.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.CALC_ELEVATION_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Calculate,
    DL_Status,
    DIS_Types,
    Global_Data,
    Numeric_Types;

procedure Check_Elevation is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  Entitiy_Location : DIS_Types.A_WORLD_COORDINATE
                                  := (X => 2_000.0,
                                      Y => 4_025.0,
                                      Z => 1_025.0);

(Assume that a reference Entity State PDU with the orientation and location fields initialized is
known);

  Reference_PDU    : DIS_Types.AN_ENTITY_STATE_PDU
                                  := Global_Data.Entity_State_PDU;

  The_Elevation   : Numeric_Types.FLOAT_32_BIT;

  .
  .

begin

  .
  .
  -- Get elevation
  Calculate.Elevation (
    Entity_State_PDU     => Reference_PDU,
    Position_Of_Interest => Entity_Location,
    Elevation            => The_Elevation,
    Status               => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
     end Check_Elvation;
```

Calculate.Velocity Unit tc "Calculate.Velocity Unit"\l 4§

Procedure Syntax:

```
Calculate.Velocity (
  Linear_Velocity    : in     DIS_Types.A_VECTOR;
  Velocity_Magnitude :    out Numeric_Types.FLOAT_32_BIT;
  Status             :    out DL_Status.STATUS_TYPE)
```

Remarks:

Calculates the magnitude of an input linear velocity vector using the following formula:

µ §

where:

- *m* is the magnitude.

µSYMBOL 183 \f "Symbol" \s 10 \h§ are the components of the linear velocity vector.

§

Input Parameters:

Linear_Velocity - Specifies the velocity of an entity in terms of it's x, y, z components in meters per second.

Output Parameters:

Velocity_Magnitude - The magnitude of the entity's velocity vector in meters per second.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.CALC_VELOCITY_FAILURE - Indicates an exception was raised in this unit.

Example:

```
with Calculate,
    DL_Status,
    DIS_Types,
    Numeric_Types;

procedure Check_Velocity is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  The_Velocity    : Numeric_Types.FLOAT_32_BIT;

  Velocity_Vector : DIS_Types.A_VECTOR
                                  := (X => 400.0,
                                      Y => 425.0,
                                      Z => -10.0);
  .
  .

begin

  .
  .
  -- Get the magnitude of the velocity vector.
  Calculate.Velocity (
    Linear_Velocity    => Velocity_Vector
    Velocity_Magnitude => The_Velocity,
   Status              => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Check_Velocity;
```

Coordinate_Conversions.Entity_To_Geocentric_Conversion Unit tc "Coordinate_Conversions.Entity_To_Geocentric_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Entity_To_Geocentric_Conversion(
  Euler_Angles          : in      DIS_Types.
                                                  AN_EULER_ANGLES_RECORD;
  Entity_Coordinate_
    System_Center       : in      DIS_Types.A_WORLD_COORDINATE;
  Entity_Coordinates    : in      DIS_Types.
                                                  AN_ENTITY_COORDINATE_VECTOR;
  Geocentric_Coordinates :    out DIS_Types.A_WORLD_COORDINATE;
  Status                :     out DL_Status.STATUS_TYPE);
```

Remarks:

Converts from the entity of interest coordinate system to a geocentric coordinate system.

Input Parameters:

Euler_Angles - Defines the reference entity's location in the geocentric coordinate system in terms of pitch (rotation about the lateral or X axis), roll (rotation about the longitudinal or Y axis), and yaw (rotation about the altitude or Z axis). Angles are defined in degrees.

Entity_Coordinate_System_Center - The origin of the entity of interest coordinate system defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Entity_Coordinates - The location of the point of interest defined in terms of the x, y, z axis in the reference entity's coordinate system in meters.

Output Parameters:

Geocentric_Coordinates - The entity of interest location defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.ENT_GCC_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will

terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Coordinate_Conversion,
    DL_Status,
    DIS_Types;

procedure Convert_Coordinates is

   -- Declare local variables

   Call_Status            : DL_Status.STATUS_TYPES;

   Coordinate_System_Origin : DIS_Types.A_WORLD_COORDINATE
                                              := (X => 2_500.0,
                                                  Y => 6_000.0,
                                                  Z => 3_500.0);

   Entity_Location        : DIS_Types.
                                              AN_ENTITY_COORDINATE_VECTOR;
                                               := (X => 5_500.0,
                                                   Y => 6_500.0,
                                                   Z => 4_500.0);

   Entity_Orientation     : DIS_Types.AN_EULER_ANGLES_RECORD
                                              := (Psi   => 75.0,
                                                  Theta => 50.0,
                                                  Phi   => 45.0);

   Geocentric_Location    : DIS_Types.A_WORLD_COORDINATE;
   .
   .
begin
   .
   .
   -- Convert the entity coordinates to geocentric coordinates.
Coordinate_Conversion.Entity_To_Geocentric_Conversion(
  Euler_Angles                 => Entity_Orientation,
  Entity_Coordinate_System_Center => Coordinate_System_Origin,
  Entity_Coordinates           => Entity_Location,
  Geocentric_Coordinates       => Geocentric_Location,
  Status                       => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
   end if;
   .
   .
    end Convert_Coordinates;
```

Coordinate_Conversion.Entity_To_Geocentric_Vel_Conversion Unit tc "Coordinate_Conversion.Entity_To_Geocentric_Vel_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Entity_To_Geocentric_Vel_Conversion(
  Euler_Angles          : in     DIS_Types.
                                                  AN_EULER_ANGLES_RECORD;
  Entity_Coordinate_
    System_Center       : in     DIS_Types.A_WORLD_COORDINATE;
  Geocentric_Coordinates : in    DIS_Types.
                                                  A_LINEAR_VELOCITY_VECTOR;
  Entity_Coordinates    :    out DIS_Types.
                                                  A_LINEAR_VELOCITY_VECTOR;

  Status                :    out DL_Status.STATUS_TYPE);
```

Remarks:

This procedure is the same as Coordinate_Conversion.Entity_To_Geocentric_ Conversion with the exception that the Entity_Coordinates and Geocentric_Coordinates parameter's data types are changed to accommodate the conversion of a velocity vector.

Coordinate_Conversion.Geocentric_To_Entity_Conversion Unit tc "Coordinate_Conversion.Geocentric_To_Entity_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Geocentric_To_Entity_Conversion(
  Euler_Angles          : in     DIS_Types.
                                                  AN_EULER_ANGLES_RECORD;
  Entity_Coordinate_
    System_Center       : in     DIS_Types.A_WORLD_COORDINATE;
  Geocentric_Coordinates : in    DIS_Types.A_WORLD_COORDINATE;
  Entity_Coordinates    :    out DIS_Types.
                                                  AN_ENTITY_COORDINATE_VECTOR;
  Status                :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts from a geocentric coordinate system to the entity of interest coordinate system.

Input Parameters:

Euler_Angles - Defines the reference entity's location in the geocentric coordinate system in terms of pitch (rotation about the lateral or X axis), roll (rotation about the longitudinal or Y axis), and yaw (rotation about the altitude or Z axis). Angles are defined in degrees.

Entity_Coordinate_System_Center - The origin of the entity of interest coordinate system defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Geocentric_Coordinates - The entity of interest location defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Output Parameters:

Entity_Coordinates - The location of the point of interest defined in terms of the x, y, z axis in the reference entity's coordinate system in meters.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.GCC_ENT_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Coordinate_Conversion,
   DL_Status,
   DIS_Types;

procedure Convert_Coordinates is

  -- Declare local variables

  Call_Status           : DL_Status.STATUS_TYPES;

  Coordinate_System_Origin : DIS_Types.A_WORLD_COORDINATE
                                        := (X => 2_500.0,
                                            Y => 6_000.0,
                                            Z => 3_500.0);

  Entity_Location       : DIS_Types.
                                        AN_ENTITY_COORDINATE_VECTOR;

  Entity_Orientation    : DIS_Types.AN_EULER_ANGLES_RECORD
                                        := (Psi   => 75.0,
                                            Theta => 50.0,
                                            Phi   => 45.0);

  Geocentric_Location   : DIS_Types.A_WORLD_COORDINATE;
                                        := (X => 5_500.0,
                                            Y => 6_500.0,
                                            Z => 4_500.0);
  .
begin
  .
  -- Convert the geocentric coordinates to entity coordinates.
  Coordinate_Conversion.Entity_To_Geocentric_Conversion(
    Euler_Angles                => Entity_Orientation,
    Entity_Coordinate_System_Center => Coordinate_System_Origin,
    Geocentric_Coordinates      => Geocentric_Location,
    Entity_Coordinates          => Entity_Location,
    Status                      => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
  end if;
  .
    end Convert_Coordinates;
```

Coordinate_Conversion.Geocentric_To_Entity_Vel_Conversion Unit tc "Coordinate_Conversion. Geocentric_To_Entity_Vel_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Geocentric_To_Entity_Vel_Conversion(
  Euler_Angles          : in     DIS_Types.
                                                  AN_EULER_ANGLES_RECORD;
  Entity_Coordinate_
    System_Center       : in     DIS_Types.A_WORLD_COORDINATE;
  Geocentric_Coordinates : in    DIS_Types.
                                                  A_LINEAR_VELOCITY_VECTOR;
  Entity_Coordinates    :    out DIS_Types.
                                                  A_LINEAR_VELOCITY_VECTOR;
  Status                :    out DL_Status.STATUS_TYPE);
```

Remarks:

This procedure is the same as Coordinate_Conversion.Geocentric_To_Entity_ Conversion with the exception that the Entity_Coordinates and Geocentric_Coordinates parameter's data types are changed to accommodate the conversion of a velocity vector.

Coordinate_Conversion.Geocentric_To_Geodetic_Conversion Unit tc "Coordinate_Conversion.Geocentric_To_Geodetic_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Geocentric_To_Geodetic_Conversion(
  Geocentric_Coordinates : in     DIS_Types.A_WORLD_COORDINATE;
  Geodetic_Coordinates   :    out DIS_Types.
                                                  THE_GEODETIC_COORDINATES;
  Status                 :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts geocentric x, y, z coordinates to geodetic latitude, longitude, and altitude.

Input Parameters:

Geocentric_Coordinates - Entity location defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Output Parameters:

Geodetic_Coordinates - Position coordinates defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.GCC_GDC_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Coordinate_Conversion,
    DL_Status,
    DIS_Types;

procedure Convert_Coordinates is

  -- Declare local variables

  Call_Status        : DL_Status.STATUS_TYPES;

  Geocentric_Location : DIS_Types.A_WORLD_COORDINATE;
                                          := (X => 5_500.0,
                                              Y => 6_500.0,
                                              Z => 4_500.0);

  Geodetic_Location   : DIS_Types.
                                        AN_ENTITY_COORDINATE_VECTOR;

  .
  .

begin

  .
  .
  -- Convert geocentric coordinates to geodetic coordinates.
  Coordinate_Conversion.Geocentric_To_Geodetic_Conversion(
    Geocentric_Coordinates => Geocentric_Location,
    Geodetic_Coordinates   => Geodetic_Location,
    Status                 => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Convert_Coordinates;
```

Coordinate_Conversion.Geocentric_To_Local_ConversionUnit tc "Coordinate_Conversion.Geocentric_To_Local_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Geocentric_To_Local_Conversion(
  Geocentric_Coordinates : in     DIS_Types.A_WORLD_COORDINATE;
  Local_Origin           : in     DL_Types.THE_LOCAL_ORIGIN;
  Local_Coordinates      :    out DL_Types.THE_LOCAL_COORDINATES;
  Status                 :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts geocentric coordinates to local coordinates.

Input Parameters:

Geocentric_Coordinates - Entity location defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Local_Origin - Origin of local coordinates defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Output Parameters:

Local_Coordinates - Entity location defined in terms of the x, y, z axis in the local coordinate system in feet.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.GCC_LOC_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Coordinate_Conversion,
    DL_Status,
    DL_Types,
    DIS_Types;

procedure Convert_Coordinates is

  -- Declare local variables

  Call_Status           : DL_Status.STATUS_TYPES;

  Geocentric_Location    : DIS_Types.A_WORLD_COORDINATE;
                                          := (X => 5_500.0,
                                              Y => 6_500.0,
                                              Z => 4_500.0);

  Local_Location        : DL_Types.THE_LOCAL_COORDINATES;

  Origin_Of_Local_System : DIS_Types.DL_Types.THE_LOCAL_ORIGIN
                                        := (Latitude  =>    60.0
                                            Longitude =>   115.0
                                            Altitude  => 4_500.0);
  .
  .

begin

  .
  .
  -- Convert geocentric coordinates to local coordinates.
  Coordinate_Conversion.Geocentric_To_Local_Conversion(
    Geocentric_Coordinates => Geocentric_Location,
    Local_Origin          => Origin_Of_Local_System,
    Local_Coordinates     => Local_Location,
    Status                => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Convert_Coordinates;
```

Coordinate_Conversion.Geocentric_To_Local_In_Meters_Conversion Unit tc "Coordinate_Conversion.Geocentric_To_Local_In_Meters_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Geocentric_To_Local_In_Meters_Conversion(
  Geocentric_Coordinates : in     DIS_Types.A_WORLD_COORDINATE;
  Local_Origin           : in     DL_Types.THE_LOCAL_ORIGIN;
  Local_Coordinates      :    out DL_Types.THE_LOCAL_COORDINATES;
  Status                 :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts geocentric coordinates to local coordinates.

Note: This unit is the same as Geocentric_To_Local_Conversion except the local coordinates returned are in meters instead of feet.

Input Parameters:

Geocentric_Coordinates - Entity location defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Local_Origin - Origin of local coordinates defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Output Parameters:

Local_Coordinates - Entity location defined in terms of the x, y, z axis in the local coordinate system in meters.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.GCC_LOC_M_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

See Geocentric_To_Local_Conversion

Coordinate_Conversion.Geodetic_To_Geocentric Unit tc "
Coordinate_Conversion.Geodetic_To_Geocentric Unit"§

Procedure Syntax:

```
Coordinate_Conversion.Geodetic_To_Geocentric_Conversion(
  Geodetic_Coordinates   : in     DL_Types.
                                                  THE_GEODETIC_COORDINATES;
  Geocentric_Coordinates :    out DIS_Types.A_WORLD_COORDINATE;
  Status                 :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts geodetic latitude, longitude, and altitude to geocentric x, y, z coordinates.

Input Parameters:

Geodetic_Coordinates - Entity location defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Output Parameters:

Geocentric_Coordinates - Entity location defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.GDC_GCC_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Coordinate_Conversion,
    DL_Status,
    DL_Types,
    DIS_Types;

procedure Convert_Coordinates is

  -- Declare local variables

  Call_Status        : DL_Status.STATUS_TYPES;

  Geocentric_Location : DIS_Types.A_WORLD_COORDINATE;

  Geodetic_Location   : DL_Types.The_GEODETIC_COORDINATES
                                        := (Latitude  =>    80.0
                                            Longitude =>   145.0
                                            Altitude  => 6_500.0);

  .
  .

begin

  .
  .
  -- Convert geodetic coordinates to  geocentric coordinates.
  Coordinate_Conversion.Geodetic_To_Geocentric_Conversion(
    Geodetic_Coordinates   => Geodetic_Location,
    Geocentric_Coordinates => Geocentric_Location,
    Status                 => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
  .
    end Convert_Coordinates;
```

Coordinate_Conversion.Geodetic_To_Local_Conversion Unit tc "Coordinate_Conversion.Geodetic_To_Local_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Geodetic_To_Local_Conversion(
  Geodetic_Coordinates : in     DL_Types.
                                          THE_GEODETIC_COORDINATES;
  Local_Origin         : in     DL_Types.THE_LOCAL_ORIGIN;
  Local_Coordinates    :    out DL_Types.THE_LOCAL_COORDINATES;
  Status               :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts geodetic latitude, longitude, and altitude to local x, y, z coordinates.

Input Parameters:

Geodetic_Coordinates - Entity location defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Local_Origin - Origin of local coordinates defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Output Parameters:

Local_Coordinates - Entity location defined in terms of the x, y, z axis in the local coordinate system in feet.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.GDC_LOC_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Coordinate_Conversion,
   DL_Status,
   DL_Types;

procedure Convert_Coordinates is

  -- Declare local variables

  Call_Status          : DL_Status.STATUS_TYPES;

  Geodetic_Location    : DL_Types.The_GEODETIC_COORDINATES
                                        := (Latitude  =>   80.0
                                            Longitude =>  145.0
                                            Altitude  => 6_500.0);

  Local_Location       : DL_Types.THE_LOCAL_COORDINATES;

  Origin_Of_Local_System : DL_Types.THE_LOCAL_ORIGIN
                                        := (Latitude  => 60.0
                                            Longitude => 115.0
                                            Altitude  => 4_500.0);

  .
  .

begin

  .
  .
  -- Convert geodetic coordinates to local coordinates.
  Coordinate_Conversion.Geodetic_To_Local_Conversion(
    Geodetic_Coordinates => Geodetic_Location,
    Local_Origin         => Origin_Of_Local_System,
    Local_Coordinates    => Local_Location,
    Status               => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Convert_Coordinates;
```

Coordinate_Conversion.Local_To_Geocentric_Conversion Unit tc "Coordinate_Conversion.Local_To_Geocentric_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Local_To_Geocentric_Conversion(
  Local_Coordinates     : in     DL_Types.THE_LOCAL_COORDINATES;
  Local_Origin          : in     DL_Types.THE_LOCAL_ORIGIN;
  Geocentric_Coordinates :    out DIS_Types.A_WORLD_COORDINATE;
  Status                :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts local coordinates to geocentric coordinates.

Input Parameters:

Local_Coordinates - Entity location defined in terms of the x, y, z axis in the local coordinate system in feet.

Local_Origin - Origin of local coordinates defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Output Parameters:

Geocentric_Coordinates - Entity location defined in terms of the x, y, z axis in the geocentric coordinate system in meters.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.LOC_GCC_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Coordinate_Conversion,
   DL_Types,
   DL_Status,
   DIS_Types;

procedure Convert_Coordinates is

  -- Declare local variables

  Call_Status          : DL_Status.STATUS_TYPES;

  Geocentric_Location    : DIS_Types.A_WORLD_COORDINATE;

  Local_Location        : DL_Types.THE_LOCAL_COORDINATES
                                          := (X => 5_500.0,
                                              Y => 6_500.0,
                                              Z => 4_500.0);

  Origin_Of_Local_System : DL_Types.THE_LOCAL_ORIGIN
                                         := (Latitude  =>   60.0
                                             Longitude =>  115.0
                                             Altitude  => 4_500.0);

  .
  .

begin

  .
  .
  -- Convert local coordinates to geocentric coordinates.
  Coordinate_Conversion.Local_To_Geocentric_Conversion(
    Local_Coordinates     => Local_Location,
    Local_Origin          => Origin_Of_Local_System,
    Geocentric_Coordinates => Geocentric_Location,
    Status                => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
  (call encountered an error - handle error)
  .
  .
  end if;
  .
  .
    end Convert_Coordinates;
```

Coordinate_Conversion.Local_To_Geodetic_Conversion Unit tc "Coordinate_Conversion.Local_To_Geodetic_Conversion Unit"\l 4§

Procedure Syntax:

```
Coordinate_Conversion.Local_To_Geodetic_Conversion(
  Local_Coordinates    : in     DL_Types.THE_LOCAL_COORDINATES;
  Local_Origin         : in     DL_Types.THE_LOCAL_ORIGIN;
  Geodetic_Coordinates :    out DIS_Types.
                                              THE_GEODETIC_COORDINATES;
  Status               :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts local x, y, z coordinates to geodetic latitude, longitude, and altitude.

Input Parameters:

Local_Coordinates - Entity location defined in terms of the x, y, z axis in the local coordinate system in feet.

Local_Origin - Origin of local coordinates defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Output Parameters:

Geodetic_Coordinates - Entity location defined in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.LOC_GDC_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Coordinate_Conversion,
   DL_Status,
   DL_Types,
   DIS_Types;

procedure Convert_Coordinates is

  -- Declare local variables

  Call_Status          : DL_Status.STATUS_TYPES;

  Geodetic_Location    : DIS_Types.A_WORLD_COORDINATE;

  Local_Location       : DL_Types.THE_LOCAL_COORDINATES
                                    := (X => 3_500.0,
                                        Y => 1_450.0,
                                        Z => 6_500.0);

  Origin_Of_Local_System : DL_Types.THE_LOCAL_ORIGIN
                                      := (Latitude  =>   60.0
                                          Longitude =>  115.0
                                          Altitude  => 4_500.0);

  .
  .

begin

  .
  .
  -- Convert local coordinates to geodetic coordinates.
  Coordinate_Conversion.Local_To_Geodetic_Conversion(
    Local_Coordinates    => Local_Location,
    Local_Origin         => Origin_Of_Local_System,
    Geodetic_Coordinates => Geodetic_Location,
    Status               => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Convert_Coordinates;
```

Dead_Reckoning.Update_Position Unit tc "Dead_Reckoning.Update_Position Unit"\l 4§

Procedure Syntax:

```
Dead_Reckoning.Update_Position(
  Entity_State_PDU : in out DIS_Types.AN_ENTITY_STATE_PDU;
  Delta_Time       : in     INTEGER;
  Status           :    out DL_Status.STATUS_TYPE);
```

Remarks:

Calculates the entity's next dead-reckoned position. The following five dead-reckoning algorithms are implemented as described in DIS: DRM(F,P,W), DRM(F,V,W), DRM(R,P,W), DRM(R,V,W) and Static. All of the implemented algorithms are for world coordinate systems. The input of any other algorithm will result in the execution of the default algorithm, Static, which means that the entity will not be moved.

Input/Output Parameters:

Entity_State_PDU - Provides all the entity information required to dead reckon the entity's position.

Input Parameters:

Delta_Time - The time (in microseconds) that has elapsed since the position values were last assigned.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.DR_UDRP_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with Dead_Reckoning,
    DL_Status,
    DIS_Types,
    Global_Data;

procedure Dead_Reckon_Entity is

  -- Declare local variables

  Call_Status      : DL_Status.STATUS_TYPES;

(Assume that the following information is known: an Entity State PDU with the kinematic data
and dead-reckoning algorithm initialized, current time and time of last update.)

  Elapsed_Time     : INTEGER
                                    := Global_Data.Current_Time - Global_Data.Last_Time;

  Entity_State_PDU : DIS_Types.AN_ENTITY_STATE_PDU
                                  := Global_Data.Entity_State_PDU;

  .
  .

begin

  .
  .
  --Dead-reckon the entity's position
  Dead_Reckoning.Update_Position(
    Entity_State_PDU => Entity_State_PDU,
    Delta_Time       => Elapsed_Time,
    Status           => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Dead_Reckon_Entity;
```

Filter.Az_And_EL Unit tc "Filter.Az_And_El Unit"\l 4§

Procedure Syntax:

```
Filter.Az_And_El(
  Az_Angle_1          : in     Numeric_Types.FLOAT_32_BIT;
  Az_Angle_2          : in     Numeric_Types.FLOAT_32_BIT;
  El_Angle_1          : in     Numeric_Types.FLOAT_32_BIT;
  El_Angle_2          : in     Numeric_Types.FLOAT_32_BIT;
  Entity_State_PDU    : in     DIS_Types.AN_ENTITY_STATE_PDU;
  Position_Of_Interest : in    DIS_Types.A_WORLD_COORDINATE;
  Status              :    out DL_Status.STATUS_TYPE;
  Within_Threshold    :    out BOOLEAN);
```

Remarks:

Determines whether the azimuth and elevation of an entity with respect to a reference entity are within the specified threshold ranges.

Note: This unit is the same as Filter_By_PDU_Component.Az_And_El except the input of the entity's Location and Orientation in that unit has been replaced with the Entity_State_PDU in this unit.

Input Parameters:

Az_Angle_1 - Specifies the first angle in degrees of the azimuth threshold range.

Az_Angle_2 - Specifies the second angle in degrees of the azimuth threshold range.

El_Angle_1 - Specifies the first angle in degrees of the elevation threshold range.

El_Angle_2 - Specifies the second angle in degrees of the elevation threshold range.

Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Position_Of_Interest - Holds a geocentric position in meters which is translated into the entity coordinate system of the entity described by the Entity_State_PDU.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.

- DL_Status.FILT_AZ_AND_EL_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Within_Threshold - Set to TRUE if both the elevation and azimuth angles are within the specified thresholds; otherwise, it is set to FALSE.

Example:

```
with DL_Status,
   DIS_Types,
   Filter,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  Az_Threshold_1  : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2  : Numeric_Types.FLOAT_32_BIT := 65.0
  El_Threshold_1  : Numeric_Types.FLOAT_32_BIT := 20.0
  El_Threshold_2  : Numeric_Types.FLOAT_32_BIT := 40.0;

  Entity_Location : DIS_Types.A_WORLD_COORDINATE
                                  :=(X => 2_000.0,
                                     Y => 4_025.0,
                                     Z => 1_025.0);

  Keep_PDU        : BOOLEAN;

(Assume that a reference Entity State PDU with the orientation and location fields initialized is
known);

  Reference_PDU   : DIS_Types.AN_ENTITY_STATE_PDU
                                  := Global_Data.Entity_State_PDU;
  .

begin
  .

  --Check to see if the PDU meets the specified criteria.
  Filter.Az_And_El(
    Az_Angle_1          => Az_Threshold_1,
    Az_Angle_2          => Az_Threshold_2,
    El_Angle_1          => El_Threshold_1,
    El_Angle_2          => El_Threshold_2,
    Entity_State_PDU    => Reference_PDU,
    Position_Of_Interest => Entity_Location,
    Status              => Call_Status,
    Within_Threshold    => Keep_PDU);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
  end if;
  .
    end Filter_PDU;
```

Filter.Azimuth Unit tc "Filter.Azimuth Unit"\l 4§

Procedure Syntax:

```
Filter.Azimuth(
  Angle_1           : in     Numeric_Types.FLOAT_32_BIT;
  Angle_2           : in     Numeric_Types.FLOAT_32_BIT;
  Entity_State_PDU     : in     DIS_Types.AN_ENTITY_STATE_PDU;
  Position_Of_Interest : in     DIS_Types.A_WORLD_COORDINATE;
  Status            :    out DL_Status.STATUS_TYPE;
  Within_Threshold     :    out BOOLEAN);
```

Remarks:

Determines whether the azimuth of an entity with respect to a reference entity is within the specified threshold ranges.

Input Parameters:

Angle_1 - Specifies the first angle in degrees of the azimuth threshold range.

Angle_2 - Specifies the second angle in degrees of the azimuth threshold range.

Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Position_Of_Interest - Holds a geocentric position in meters which is translated into the entity coordinate system of the entity described by the Entity_State_PDU.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_AZ_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Within_Threshold - Set to TRUE if the azimuth angle is within the specified threshold; otherwise, it is set to FALSE.

Example:

```
with DL_Status,
   DIS_Types,
   Filter,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  Az_Threshold_1  : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2  : Numeric_Types.FLOAT_32_BIT := 65.0


  Entity_Location : DIS_Types.A_WORLD_COORDINATE
                                  :=(X => 2_000.0,
                                     Y => 4025.0,
                                     Z => 1_025.0);

  Keep_PDU        : BOOLEAN;

(Assume that a reference Entity State PDU with the orientation and location fields initialized is
known);

  Reference_PDU   : DIS_Types.AN_ENTITY_STATE_PDU
                                  := Global_Data.Entity_State_PDU;
  .

begin

  .
  .
  --Check to see if the PDU meets the specified criteria.
  Filter.Azimuth(
    Angle_1            => Az_Threshold_1,
    Angle_2            => Az_Threshold_2,
    Entity_State_PDU   => Reference_PDU,
    Position_Of_Interest => Entity_Location,
    Status             => Call_Status,
    Within_Threshold   => Keep_PDU);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
  .
    end Filter_PDU;
```

Filter.Distance Unit tc "Filter.Distance Unit"\l 4§

Procedure Syntax:

```
Filter.Distance(
  Position_1      : in    DIS_Types.A_WORLD_COORDINATE;
  Position_2      : in    DIS_Types.A_WORLD_COORDINATE;
  Threshold       : in    Numeric_Types.FLOAT_64_BIT;
  Status          :   out DL_Status.STATUS_TYPE;
  Within_Threshold :   out BOOLEAN);
```

Remarks:

Determines whether the distance between two entities is within a specified maximum distance threshold value.

Input Parameters:

Position_1 - A vector position of an entity in meters.

Position_2 - A vector position of an entity in meters.

Threshold - The maximum distance between the entities in meters.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_DIST_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Within_Threshold - Set to TRUE if the distance between the two input positions is less than or equal to the maximum distance threshold value; otherwise, it is set to FALSE.

Example:

```
with DL_Status,
    DIS_Types,
    Filter,
    Numeric_Types;

procedure Filter_PDU is

  -- Declare local variables

  Call_Status : DL_Status.STATUS_TYPES;

  Entity_1    : DIS_Types.A_WORLD_COORDINATE;
                              :=(X => 2_000.0,
                                 Y => 4025.0,
                                 Z => 1_025.0);

  Entity_2    : DIS_Types.A_WORLD_COORDINATE;
                               :=(X => 1_000.0,
                                  Y => 3025.0,
                                  Z => 1_025.0);

  Keep_PDU        : BOOLEAN;

  Maximum_Distance : Numeric_Types.FLOAT_64_BIT := 1_500.0

  .
  .

begin

  .
  .
  --Check to see if the PDU meets the specified criteria.
  Filter.Distance(
    Position_1       => Entity_1,
    Position_2       => Entity_2,
    Threshold        => Maximum_Distance,
    Status           => Call_Status,
    Within_Threshold => Keep_PDU);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
  .
    end Filter_PDU;
```

Filter.Elevation Unit tc "Filter.Elevation Unit"\l 4§

Procedure Syntax:

```
Filter.Elevation(
  Angle_1           : in    Numeric_Types.FLOAT_32_BIT;
  Angle_2           : in    Numeric_Types.FLOAT_32_BIT;
  Entity_State_PDU    : in    DIS_Types.AN_ENTITY_STATE_PDU;
  Position_Of_Interest : in    DIS_Types.A_WORLD_COORDINATE;
  Status            :   out DL_Status.STATUS_TYPE;
  Within_Threshold    :   out BOOLEAN);
```

Remarks:

Determines whether the elevation of an entity with respect to a reference entity are within the specified threshold ranges.

Input Parameters:

Angle_1 - Specifies the first angle in degrees of the elevation threshold range.

Angle_2 - Specifies the second angle in degrees of the elevation threshold range.

Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Position_Of_Interest - Holds a geocentric position in meters which is translated into the entity coordinate system of the entity described by the Entity_State_PDU.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_EL_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Within_Threshold - Set to TRUE if the elevation angle is within the specified threshold; otherwise, it is set to FALSE.

Example:

```
with DL_Status,
   DIS_Types,
   Filter,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  El_Threshold_1  : Numeric_Types.FLOAT_32_BIT := 20.0
  El_Threshold_2  : Numeric_Types.FLOAT_32_BIT := 40.0;

  Entity_Location : DIS_Types.A_WORLD_COORDINATE
                                  :=(X => 2_000.0,
                                     Y => 4025.0,
                                     Z => 1_025.0);

  Keep_PDU        : BOOLEAN;

(Assume that a reference Entity State PDU with the orientation and location fields initialized is
known);

  Reference_PDU   : DIS_Types.AN_ENTITY_STATE_PDU
                                  := Global_Data.Entity_State_PDU;
  .
  .

begin

  .
  .
  --Check to see if the PDU meets the specified criteria.
  Filter.Elevation(
    Angle_1            => El_Threshold_1,
    Angle_2            => El_Threshold_2,
    Entity_State_PDU   => Reference_PDU,
    Position_Of_Interest => Entity_Location,
    Status             => Call_Status,
    Within_Threshold   => Keep_PDU);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
  .
    end Filter_PDU;
```

Filter.Maximum_Velocity Unit tc "Filter.Maximum_Velocity Unit"\l 4§

Procedure Syntax:

```
Filter.Maximum_Velocity(
  Threshold       : in     Numeric_Types.FLOAT_32_BIT;
  Velocity        : in     DIS_Types.A_VECTOR;
  Status          :    out DL_Status.STATUS_TYPE;
  Within_Threshold :    out BOOLEAN);
```

Remarks:

Determines whether the velocity magnitude of an entity is within the specified maximum velocity threshold.

Input Parameters:

Threshold - The maximum velocity in meters per second.

Velocity - Specifies the velocity of an entity in terms of it's x, y, z components in meters per second.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_MAX_VEL_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Within_Threshold - Set to TRUE if the velocity is equal to or less than the maximum velocity threshold value; otherwise, it is set to FALSE.

Example:

```
with DL_Status,
    DL_Types,
    Filter,
    Numeric_Types;

procedure Filter_PDU is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

  Keep_PDU          : BOOLEAN;

  Maximum_Velocity   : Numeric_Types.FLOAT_32_BIT := 500.0;

  Velocity_Vector    : DL_Types.THE_LOCAL_COORDINATES
                                    := (X => 200.0,
                                        Y => 250.0,
                                        Z =>  30.0);
  .
  .

begin

  .
  .
  --Check to see if the PDU meets the specified criteria.
  Filter.Maximum_Velocity(
    Threshold        => Maximum_Velocity,
    Velocity         => Velocity_Vector,
    Status           => Call_Status,
    Within_Threshold => Keep_PDU);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Filter_PDU;
```

Filter.Minimum_Velocity Unit tc "Filter.Minimum_Velocity Unit"\l 4§

Procedure Syntax:

```
Filter.Minimum_Velocity(
  Threshold       : in     Numeric_Types.FLOAT_32_BIT;
  Velocity        : in     DIS_Types.A_VECTOR;
  Status          :    out DL_Status.STATUS_TYPE;
  Within_Threshold :   out BOOLEAN);
```

Remarks:

Determines whether the velocity magnitude of an entity is within the specified minimum velocity threshold.

Input Parameters:

Threshold - The minimum velocity in meters per second.

Velocity - Specifies the velocity of an entity in terms of it's x, y, z components in meters per second.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_MIN_VEL_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Within_Threshold - Set to TRUE if the velocity is equal to or greater than the minimum velocity threshold value; otherwise, it is set to FALSE.

Example:

```
with DL_Status,
    DL_Types,
    Filter,
    Numeric_Types;

procedure Filter_PDU is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  Keep_PDU        : BOOLEAN;

  Minimum_Velocity : Numeric_Types.FLOAT_32_BIT := 200.0;

  Velocity_Vector  : DL_Types.THE_LOCAL_COORDINATES
                                  := (X => 200.0,
                                      Y => 250.0,
                                      Z =>  30.0);

  .
  .

begin

  .
  .
  --Check to see if the PDU meets the specified criteria.
  Filter.Minimum_Velocity(
    Threshold       => Minimum_Velocity,
    Velocity        => Velocity_Vector,
    Status          => Call_Status,
    Within_Threshold => Keep_PDU);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Filter_PDU;
```

Filter_By_PDU_Components.Az_And_El Unit tc "Filter_By_PDU_Components.Az_And_El Unit"\l 4§

Procedure Syntax:

```
Filter_By_PDU_Components.Az_And_El(
  Az_Angle_1          : in     Numeric_Types.FLOAT_32_BIT;
  Az_Angle_2          : in     Numeric_Types.FLOAT_32_BIT;
  El_Angle_1          : in     Numeric_Types.FLOAT_32_BIT;
  El_Angle_2          : in     Numeric_Types.FLOAT_32_BIT;
  Location            : in     DIS_Types.A_WORLD_COORDINATE;
  Orientation         : in     DIS_Types.AN_EULER_ANGLES_RECORD;
  Position_Of_Interest : in    DIS_Types.A_WORLD_COORDINATE;
  Status              :    out DL_Status.STATUS_TYPE;
  Within_Threshold    :    out BOOLEAN);
```

Remarks:

Determines whether the azimuth and elevation of an entity with respect to a reference entity are within the specified threshold ranges.

Note: This unit is the same as Filter.Az_And_El except the input parameter of Entity_State_PDU in that unit has been replaced with the entity's Location and Orientation in this unit.

Input Parameters:

Az_Angle_1 - Specifies the first angle in degrees of the azimuth threshold range.

Az_Angle_2 - Specifies the second angle in degrees of the azimuth threshold range.

El_Angle_1 - Specifies the first angle in degrees of the elevation threshold range.

El_Angle_2 - Specifies the second angle in degrees of the elevation threshold range.

Location - Reference entity's geocentric position in meters.

Orientation - Reference entity's orientation as defined by Euler Angles.

Position_Of_Interest - Holds a geocentric position in meters which is translated into the entity coordinate system of the entity described by the Entity_State_PDU.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_PDU_AZ_AND_EL_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Within_Threshold - Set to TRUE if both the elevation and azimuth angles are within the specified thresholds; otherwise, it is set to FALSE.

Example:

```
with DL_Status,
   DIS_Types,
   Filter_By_PDU_Components,
   Numeric_Types;

procedure Filter_PDU is

  -- Declare local variables

  Call_Status         : DL_Status.STATUS_TYPES;
  Az_Threshold_1      : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2      : Numeric_Types.FLOAT_32_BIT := 65.0
  El_Threshold_1      : Numeric_Types.FLOAT_32_BIT := 20.0
  El_Threshold_2      : Numeric_Types.FLOAT_32_BIT := 40.0;

  Entity_Location     : DIS_Types.A_WORLD_COORDINATE
                                    := (X : 3_500.0,
                                        Y : 4_000.0,
                                        Z : 2_000.0);

  Keep_PDU            : BOOLEAN;

  Reference_Location  : DIS_Types.A_WORLD_COORDINATE
                                    := (X : 2_500.0,
                                        Y : 2_000.0,
                                        Z : 1_000.0);

  Reference_Orientation : DIS_Types.AN_EULER_ANGLES_RECORD
                                    := (Psi   => 75.0,
                                        Theta => 50.0,
                                        Phi   => 45.0);
begin

  --Check to see if the PDU meets the specified criteria.
  Filter_By_PDU_Components.Az_And_El(
    Az_Angle_1          => Az_Threshold_1,
    Az_Angle_2          => Az_Threshold_2,
    El_Angle_1          => El_Threshold_1,
    El_Angle_2          => El_Threshold_2,
    Location          => Reference_Location,
    Orientation         => Reference_Orientation,
    Position_Of_Interest => Entity_Location,
    Status            => Call_Status,
    Within_Threshold    => Keep_PDU);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
     .
  end if;
  .
    end Filter_PDU;
```

Filter_List.Entity_State_Az_And_El.Filter_Az_And_El Unit tc "Filter_List.Entity_State_Az_And_El.Filter_Az_And_El Unit"\l 4§

Procedure Syntax:

```
Filter_List.Entity_State_Az_And_El.Filter_Az_And_El(
  The_List                : in out DL_Linked_List_Types.
                                                    Entity_State_List.PTR;
  First_Az_Threshold      : in     Numeric_Types.FLOAT_32_BIT;
  Second_Az_Threshold     : in     Numeric_Types.FLOAT_32_BIT;
  First_El_Threshold      : in     Numeric_Types.FLOAT_32_BIT;
  Second_El_Threshold     : in     Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in  DIS_Types.
                                                    AN_ENTITY_STATE_PDU;
  Status                  :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Entity State PDUs by azimuth and elevation.

This is an instantiation of the generic package Generic_Filter_List_By_Az_And_El and a call to the package's generic procedure Filter_Az_And_El which returns a sublist of Entity State PDUs that meet both the specified azimuth and elevation thresholds.

Note: This is an overloaded procedure. This procedure's parameter Reference_Entity_State_PDU is replaced in another version (see next) with the reference entity's location and orientation.

Input/Output Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to the Reference_Entity_State_PDU.

Input Parameters:

First_Az_Threshold - Specifies the first angle in degrees of the azimuth threshold range.

Second_Az_Threshold - Specifies the second angle in degrees of the azimuth threshold range.

First_El_Threshold - Specifies the first angle in degrees of the elevation threshold range.

Second_El_Threshold - Specifies the second angle in degrees of the elevation threshold range.

Reference_Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_AZ_EL_ENT_ST_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Filter_List,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status      : DL_Status.STATUS_TYPES;

  Az_Threshold_1   : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2   : Numeric_Types.FLOAT_32_BIT := 65.0
  El_Threshold_1   : Numeric_Types.FLOAT_32_BIT := 20.0
  El_Threshold_2   : Numeric_Types.FLOAT_32_BIT := 40.0;

(Assume that a list of Entity State PDUs and a reference Entity State PDU with the orientation
and location fields initialized are known);

  Entity_State_PDUs : DL_Linked_List_Types.Entity_State_List.PTR
                                    := Global_Data.Entity_State_PDU_List;

  Reference_PDU    : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                                    := Global_Data.Reference_Position;
  .
  .

begin

  .
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Entity_State_Az_And_El.Filter_Az_And_El(
    The_List               => Entity_State_PDUs,
    First_Az_Threshold     => Az_Threshold_1,
    Second_Az_Threshold    => Az_Threshold_2,
    First_El_Threshold     => El_Threshold_1,
    Second_El_Threshold    => El_Threshold_2,
    Reference_Entity_State_PDU => Reference_PDU,
    Status                 => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
    end Filter_PDU_List;
```

Filter_List.Entity_State_Az_And_El.Filter_Az_And_El Unit tc "Filter_List.Entity_State_Az_And_El.Filter_Az_And_El Unit"\l 4§

Procedure Syntax:

```
Filter_List.Entity_State_Az_And_El.Filter_Az_And_El(
  The_List           : in out DL_Linked_List_Types.
                                                Entity_State_List.PTR;
  First_Az_Threshold  : in     Numeric_Types.FLOAT_32_BIT;
  Second_Az_Threshold : in     Numeric_Types.FLOAT_32_BIT;
  First_El_Threshold  : in     Numeric_Types.FLOAT_32_BIT;
  Second_El_Threshold : in     Numeric_Types.FLOAT_32_BIT;
  Location           : in     DIS_Types.A_WORLD_COORDINATE;
  Orientation        : in     DIS_Types.AN_EULER_ANGLES_RECORD;
  Status             :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Entity State PDUs by azimuth and elevation.

This is an instantiation of the generic package Generic_Filter_List_By_Az_And_El and a call to the package's generic procedure Filter_Az_And_El which returns a sublist of Entity State PDUs that meet both the specified azimuth and elevation thresholds.

Note: This is an overloaded procedure. This procedure's parameters Location and Orientation are replaced by Reference_Entity_State_PDU in another version (see previous).

Input/Output Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to the Reference_Entity_State_PDU.

Input Parameters:

First_Az_Threshold - Specifies the first angle in degrees of the azimuth threshold range.

Second_Az_Threshold - Specifies the second angle in degrees of the azimuth threshold range.

First_El_Threshold - Specifies the first angle in degrees of the elevation threshold range.

Second_El_Threshold - Specifies the second angle in degrees of the elevation threshold range.

Location - Reference entity's geocentric position in meters.

Orientation - Reference entity's orientation as defined by Euler Angles in degrees.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_AZ_EL_ENT_ST_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Filter_List,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables
  Call_Status         : DL_Status.STATUS_TYPES;
  Az_Threshold_1      : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2      : Numeric_Types.FLOAT_32_BIT := 65.0
  El_Threshold_1      : Numeric_Types.FLOAT_32_BIT := 20.0
  El_Threshold_2      : Numeric_Types.FLOAT_32_BIT := 40.0;

(Assume that a list of Entity State PDUs the orientation and location fields initialized is known);
  Entity_State_PDUs   : DL_Linked_List_Types.
                                    Entity_State_List.PTR
                                        := Global_Data.
                                            Entity_State_PDU_List;

  Reference_Location  : DIS_Types.A_WORLD_COORDINATE
                                    := (X : 2_500.0,
                                        Y : 2_000.0,
                                        Z : 1_000.0);

  Reference_Orientation : DIS_Types.AN_EULER_ANGLES_RECORD
                                    := (Psi   => 75.0,
                                        Theta => 50.0,
                                        Phi   => 45.0);
begin
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Entity_State_Az_And_El.Filter_Az_And_El(
    The_List              => Entity_State_PDUs,
    First_Az_Threshold    => Az_Threshold_1,
    Second_Az_Threshold   => Az_Threshold_2,
    First_El_Threshold    => El_Threshold_1,
    Second_El_Threshold   => El_Threshold_2,
    Location              => Reference_Location,
    Orientation           => Reference_Orientation,
    Status                => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
  end if;
  .
   end Filter_PDU_List;
```

Filter_List.Detonation_Azimuth.Filter_Orientation Unit tc "Filter_List.Detonation_Azimuth.Filter_Orientation Unit"\l 4§

Procedure Syntax:

```
Filter_List.Detonation_Azimuth.Filter_Orientation(
  The_List                : in out DL_Linked_List_Types.
                                                    Detonation_List.PTR;
  Threshold_1             : in     Numeric_Types.FLOAT_32_BIT;
  Threshold_2             : in     Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in  DIS_Types.AN_DETONATION_PDU;
  Status                  :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Detonation PDUs by azimuth.

This is an instantiation of the generic package Generic_Filter_List_By_Orientation with the Orientation procedure instantiated to filter by azimuth and a call to the package's generic procedure Filter_Orientation which returns a sublist of Detonation PDUs that meet the specified azimuth threshold.

Input/Output Parameters:

The_List - A pointer to a list of Detonation PDUs which will be evaluated with respect to the Reference_Entity_State_PDU.

Input Parameters:

Threshold_1 - Specifies the first angle in degrees of the azimuth threshold range.

Threshold_2 - Specifies the second angle in degrees of the azimuth threshold range.

Reference_Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_AZ_DETON_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Filter_List,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

  Az_Threshold_1  : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2  : Numeric_Types.FLOAT_32_BIT := 65.0

(Assume that a list of Detonation PDUs and a reference Entity State PDU with the orientation
and location fields initialized are known);

  Detonation_PDUs : DL_Linked_List_Types.Detonation_List.PTR
                              := Global_Data.Detonation_PDU_List;

  Reference_PDU   : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                               := Global_Data.Reference_Position;

  .

begin
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Detonation_Azimuth.Filter_Orientation(
    The_List              => Detonation_PDUs,
    Threshold_1             => Az_Threshold_1,
    Threshold_2             => Az_Threshold_2,
    Reference_Entity_State_PDU => Reference_PDU,
    Status                => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
  end if;
  .
    end Filter_PDU_List;
```

Filter_List.Entity_State_Azimuth.Filter_Orientation Unit tc "Filter_List.Entity_State_Azimuth.Filter_Orientation Unit"\l 4§

Procedure Syntax:

```
Filter_List.Entity_State_Azimuth.Filter_Orientation(
  The_List                : in out DL_Linked_List_Types.
                                                    Entity_State_List.PTR;
  Threshold_1             : in     Numeric_Types.FLOAT_32_BIT;
  Threshold_2             : in     Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in     DIS_Types.
                                                    AN_ENTITY_STATE_PDU;
  Status                  :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Entity State PDUs by azimuth.

This is an instantiation of the generic package Generic_Filter_List_By_Orientation with the Orientation procedure instantiated to filter by azimuth and a call to the package's generic procedure Filter_Orientation which returns a sublist of Entity State PDUs that meet the specified azimuth threshold.

Input/Output Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to the Reference_Entity_State_PDU.

Input Parameters:

Threshold_1 - Specifies the first angle in degrees of the azimuth threshold range.

Threshold_2 - Specifies the second angle in degrees of the azimuth threshold range.

Reference_Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_AZ_ENT_ST_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be

returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Filter_List,
    Global_Data,
    Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status      : DL_Status.STATUS_TYPES;

  Az_Threshold_1   : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2   : Numeric_Types.FLOAT_32_BIT := 65.0

(Assume that a list of Entity State PDUs and a reference Entity State PDU with the orientation
and location fields initialized are known);

  Entity_State_PDUs : DL_Linked_List_Types.Entity_State_List.PTR
                              := Global_Data.Entity_State_PDU_List;

  Reference_PDU    : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                                  := Global_Data.Reference_Position;
  .
begin

  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Entity_State_Azimuth.Filter_Orientation(
    The_List               => Entity_State_PDUs,
    Threshold_1            => Az_Threshold_1,
    Threshold_2            => Az_Threshold_2,
    Reference_Entity_State_PDU => Reference_PDU,
    Status                 => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
  end if;
  .
    end Filter_PDU_List;
```

Filter_List.Fire_Azimuth.Filter_Orientation Unit tc "Filter_List.Fire_Azimuth.Filter_Orientation Unit"\l 4§

Procedure Syntax:

```
Filter_List.Fire_Azimuth.Filter_Orientation(
  The_List              : in out DL_Linked_List_Types.
                                                    Fire_List.PTR;
  Threshold_1           : in    Numeric_Types.FLOAT_32_BIT;
  Threshold_2           : in    Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in    DIS_Types.
                                                    AN_ENTITY_STATE_PDU;
  Status                :   out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Fire PDUs by azimuth.

This is an instantiation of the generic package Generic_Filter_List_By_Orientation with the Orientation procedure instantiated to filter by azimuth and a call to the package's generic procedure Filter_Orientation which returns a sublist of Fire PDUs that meet the specified azimuth threshold.

Input/Output Parameters:

The_List - A pointer to a list of Fire PDUs which will be evaluated with respect to the Reference_Entity_State_PDU.

Input Parameters:

Threshold_1 - Specifies the first angle in degrees of the azimuth threshold range.

Threshold_2 - Specifies the second angle in degrees of the azimuth threshold range.

Reference_Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_AZ_FIRE_FAILURE - Indicates an exception was

raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Filter_List,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status    : DL_Status.STATUS_TYPES;

  Az_Threshold_1 : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2 : Numeric_Types.FLOAT_32_BIT := 65.0

(Assume that a list of Fire PDUs and a reference Entity State PDU with the orientation and
location fields initialized are known);

  Fire_PDUs     : DL_Linked_List_Types.Fire_List.PTR
                              := Global_Data.Fire_PDU_List;

  Reference_PDU  : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                              := Global_Data.Reference_Position;
  .

begin

  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Fire_Azimuth.Filter_Orientation(
    The_List               => Fire_PDUs,
    Threshold_1              => Az_Threshold_1,
    Threshold_2              => Az_Threshold_2,
    Reference_Entity_State_PDU => Reference_PDU,
    Status                 => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
  end if;
  .
    end Filter_PDU_List;
```

Filter_List.Detonation_Distance.Filter_Distance Unit tc "Filter_List.Detonationn_Distance.Filter_Distance Unit"\l 4§

Procedure Syntax:

```
Filter_List.Detonation_Distance.Filter_Distance(
  The_List          : in out DL_Linked_List_Types.
                                                Detonation_List.PTR;
  Reference_Position : in     DIS_Types.A_WORLD_COORDINATE;
  Threshold         : in     Numeric_Types.FLOAT_64_BIT;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Detonation PDUs by distance from a reference location.

This is an instantiation of the generic package Generic_Filter_List_By_Distance and a call to the package's generic procedure Filter_Distance which returns a sublist of Detonation PDUs that meet the specified distance threshold.

Input/Output Parameters:

The_List - A pointer to a list of Detonation PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Point from which the distance is calculated for each PDU location in the list.

Threshold - Specifies the maximum acceptable distance between the PDU location and the reference position (i.e., less than or equal to).

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_DIST_DETON_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Filter_List,
    Global_Data,
    Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Detonation PDUs with location fields initialized is known);

  Detonation_PDUs    : DL_Linked_List_Types.Detonation_List.PTR
                                        := Global_Data.Detonation_PDU_List;

  Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                        := (X : 2_500.0,
                                            Y : 2_000.0,
                                            Z : 1_000.0);

  Maximum_Distance  : Numeric_Types.FLOAT_64_BIT := 4_000.0

  .
  .

begin

  .
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Detonation_Distance.Filter_Distance(
    The_List          => Detonation_PDUs,
    Reference_Position  => Reference_Location,
    Threshold          => Maximum_Distance,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Filter_PDU_List;
```

Filter_List.Entity_State_Distance.Filter_Distance Unit tc "Filter_List.Entity_State_Distance.Filter_Distance Unit"\l 4§

Procedure Syntax:

```
Filter_List.Entity_State_Distance.Filter_Distance(
  The_List          : in out DL_Linked_List_Types.
                                                Entity_State_List.PTR;
  Reference_Position : in     DIS_Types.A_WORLD_COORDINATE;
  Threshold         : in     Numeric_Types.FLOAT_64_BIT;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Entity State PDUs by distance from a reference location.

This is an instantiation of the generic package Generic_Filter_List_By_Distance and a call to the package's generic procedure Filter_Distance which returns a sublist of Entity State PDUs that meet the specified distance threshold.

Input/Output Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Point from which the distance is calculated for each entity in the list.

Threshold - Specifies the maximum acceptable distance between the entity and the reference position (i.e., less than or equal to).

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_DIST_ENT_ST_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Filter_List,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status        : DL_Status.STATUS_TYPES;

(Assume that a list of Entity State PDUs with location fields initialized is known);

  Entity_State_PDUs   : DL_Linked_List_Types.
                                        Entity_State_List.PTR
                                            := Global_Data.
                                               Entity_State_PDU_List;

  Reference_Location  : DIS_Types.A_WORLD_COORDINATE
                                        := (X : 2_500.0,
                                            Y : 2_000.0,
                                            Z : 1_000.0);

  Maximum_Distance    : Numeric_Types.FLOAT_64_BIT := 4_000.0

  .
  .

begin

  .
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Entity_State_Distance.Filter_Distance(
    The_List          => Detonation_PDUs,
    Reference_Position => Reference_Location,
    Threshold         => Maximum_Distance,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Filter_PDU_List;
```

Filter_List.Fire_Distance.Filter_Distance Unit tc
"Filter_List.Fire_Distance.Filter_Distance Unit"\l 4§

Procedure Syntax:

```
Filter_List.Fire_Distance.Filter_Distance(
  The_List          : in out DL_Linked_List_Types.Fire_List.PTR;
  Reference_Position : in     DIS_Types.A_WORLD_COORDINATE;
  Threshold         : in     Numeric_Types.FLOAT_64_BIT;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Fire PDUs by distance from a reference location.

This is an instantiation of the generic package Generic_Filter_List_By_Distance and a call to the package's generic procedure Filter_Distance which returns a sublist of Fire PDUs that meet the specified distance threshold.

Input/Output Parameters:

The_List - A pointer to a list of Fire PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Point from which the distance is calculated for each PDU location in the list.

Threshold - Specifies the maximum acceptable distance between the PDU location and the reference position (i.e., less than or equal to).

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_DIST_FIRE_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Filter_List,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Fire PDUs with location fields initialized is known);

  Fire_PDUs         : DL_Linked_List_Types.Fire_List.PTR
                                        := Global_Data.Fire_PDU_List;

  Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                        := (X : 2_500.0,
                                            Y : 2_000.0,
                                            Z : 1_000.0);

  Maximum_Distance  : Numeric_Types.FLOAT_64_BIT := 4_000.0

  .
  .

begin

  .
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Fire_Distance.Filter_Distance(
    The_List          => Fire_PDUs,
    Reference_Position => Reference_Location,
    Threshold         => Maximum_Distance,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Filter_PDU_List;
```

Filter_List.Laser_Distance.Filter_Distance Unit tc "Filter_List.Laser_Distance.Filter_Distance Unit"\l 4§

Procedure Syntax:

```
Filter_List.Laser_Distance.Filter_Distance(
  The_List          : in out DL_Linked_List_Types.Laser_List.PTR;
  Reference_Position : in     DIS_Types.A_WORLD_COORDINATE;
  Threshold         : in     Numeric_Types.FLOAT_64_BIT;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Laser PDUs by distance from a reference location.

This is an instantiation of the generic package Generic_Filter_List_By_Distance, and a call to the package's generic procedure Filter_Distance which returns a sublist of Laser PDUs that meet the specified distance threshold.

Input/Output Parameters:

The_List - A pointer to a list of Laser PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Point from which the distance is calculated for each PDU location in the list.

Threshold - Specifies the maximum acceptable distance between the PDU location and the reference position (i.e., less than or equal to).

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_DIST_LASER_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Filter_List,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Laser PDUs with location fields initialized is known);

  Laser_PDUs        : DL_Linked_List_Types.Laser_List.PTR
                                        := Global_Data.Laser_PDU_List;

  Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                        := (X : 2_500.0,
                                            Y : 2_000.0,
                                            Z : 1_000.0);

  Maximum_Distance  : Numeric_Types.FLOAT_64_BIT := 4_000.0

  .
  .

begin

  .
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Laser_Distance.Filter_Distance(
    The_List          => Laser_PDUs,
    Reference_Position => Reference_Location,
    Threshold         => Maximum_Distance,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Filter_PDU_List;
```

Filter_List.Transmitter_Distance.Filter_Distance Unit tc "Filter_List.Transmitter_Distance.Filter_Distance Unit"\l 4§

Procedure Syntax:

```
Filter_List.Transmitter_Distance.Filter_Distance(
  The_List          : in out DL_Linked_List_Types.
                                              Transmitter_List.PTR;
  Reference_Position : in     DIS_Types.A_WORLD_COORDINATE;
  Threshold         : in     Numeric_Types.FLOAT_64_BIT;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Transmitter PDUs by distance from a reference location.

This is an instantiation of the generic package Generic_Filter_List_By_Distance and a call to the package's generic procedure Filter_Distance which returns a sublist of Transmitter PDUs that meet the specified distance threshold.

Input/Output Parameters:

The_List - A pointer to a list of Transmitter PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Point from which the distance is calculated for each PDU location in the list.

Threshold - Specifies the maximum acceptable distance between the PDU location and the reference position (i.e., less than or equal to).

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_DIST_TRANS_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Filter_List,
    Global_Data,
    Numeric_Types;

procedure Filter_PDU_List is

   -- Declare local variables

   Call_Status       : DL_Status.STATUS_TYPES;

   Maximum_Distance   : Numeric_Types.FLOAT_64_BIT := 4_000.0

   Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                        := (X : 2_500.0,
                                            Y : 2_000.0,
                                            Z : 1_000.0);
(Assume that a list of Transmitter PDUs with location fields initialized is known);

   Transmitter_PDUs   : DL_Linked_List_Types.Transmitter_List.PTR
                                        := Global_Data.Transmitter_PDU_List;

   .
   .

begin

   .
   .
   --Delete PUUs which do not meet the specified criteria.
   Filter_List.Transmitter_Distance.Filter_Distance(
     The_List          => Transmitter_PDUs,
     Reference_Position => Reference_Location,
     Threshold         => Maximum_Distance,
     Status            => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
   end if;
   .
   .
     end Filter_PDU_List;
```

Filter_List.Detonation_Elevation.Filter_Orientation Unit tc "Filter_List.Detonation_Elevation.Filter_Orientation Unit"\l 4§

Procedure Syntax:

```
Filter_List.Detonation_Elevation.Filter_Orientation(
  The_List                : in out DL_Linked_List_Types.
                                                    Detonation_List.PTR;
  Threshold_1             : in     Numeric_Types.FLOAT_32_BIT;
  Threshold_2             : in     Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in  DIS_Types.AN_DETONATION_PDU;
  Status                  :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Detonation PDUs by elevation.

This is an instantiation of the generic package Generic_Filter_List_By_Orientation with the Orientation procedure instantiated to filter by elevation and a call to the package's generic procedure Filter_Orientation which returns a sublist of Detonation PDUs that meet the specified elevation threshold.

Input/Output Parameters:

The_List - A pointer to a list of Detonation PDUs which will be evaluated with respect to the Reference_Entity_State_PDU.

Input Parameters:

Threshold_1 - Specifies the first angle in degrees of the elevation threshold range.

Threshold_2 - Specifies the second angle in degrees of the elevation threshold range.

Reference_Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_EL_DETON_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Filter_List,
    Global_Data,
    Numeric_Types;

procedure Filter_PDU_List is

   -- Declare local variables

   Call_Status     : DL_Status.STATUS_TYPES;

(Assume that a list of Detonation PDUs and a reference Entity State PDU with the orientation
and location fields initialized are known);

   Detonation_PDUs : DL_Linked_List_Types.Detonation_List.PTR
                                := Global_Data.Detonation_PDU_List;

   El_Threshold_1  : Numeric_Types.FLOAT_32_BIT := 45.0
   El_Threshold_2  : Numeric_Types.FLOAT_32_BIT := 65.0

   Reference_PDU   : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                                        := Global_Data.Reference_Position;
   .
begin
   .
   .
   --Delete PUUs which do not meet the specified criteria.
   Filter_List.Detonation_Elevation.Filter_Orientation(
     The_List                => Detonation_PDUs,
     Threshold_1               => El_Threshold_1,
     Threshold_2               => El_Threshold_2,
     Reference_Entity_State_PDU => Reference_PDU,
     Status                  => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
   end if;
   .
     end Filter_PDU_List;
```

Filter_List.Entity_State_Elevation.Filter_Orientation Unit tc "Filter_List.Entity_State_Elevation.Filter_Orientation Unit"\l 4§

Procedure Syntax:

```
Filter_List.Entity_State_Elevation.Filter_Orientation(
  The_List                : in out DL_Linked_List_Types.
                                                    Entity_State_List.PTR;
  First_EL_Threshold      : in     Numeric_Types.FLOAT_32_BIT;
  Second_EL_Threshold     : in     Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in  DIS_Types.
                                                    AN_ENTITY_STATE_PDU;
  Status                  :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Entity State PDUs by elevation.

This is an instantiation of the generic package Generic_Filter_List_By_Orientation with the Orientation procedure instantiated to filter by elevation and a call to the package's generic procedure Filter_Orientation which returns a sublist of Entity State PDUs that meet the specified elevation threshold.

Input/Output Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to the Reference_Entity_State_PDU.

Input Parameters:

Threshold_1 - Specifies the first angle in degrees of the elevation threshold range.

Threshold_2 - Specifies the second angle in degrees of the elevation threshold range.

Reference_Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_EL_ENT_ST_FAILURE - Indicates an exception was

raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Filter_List,
   Global_Data,
   Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status      : DL_Status.STATUS_TYPES;

  EL_Threshold_1   : Numeric_Types.FLOAT_32_BIT := 45.0
  EL_Threshold_2   : Numeric_Types.FLOAT_32_BIT := 65.0

(Assume that a list of Entity State PDUs and a reference Entity State PDU with the orientation
and location fields initialized are known);

  Entity_State_PDUs : DL_Linked_List_Types.Entity_State_List.PTR
                                  := Global_Data.Entity_State_PDU_List;

  Reference_PDU    : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                                     := Global_Data.Reference_Position;
  .

begin
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Entity_State_Elevation.Filter_Orientation(
    The_List               => Entity_State_PDUs,
    Threshold_1              => El_Threshold_1,
    Threshold_2              => El_Threshold_2,
    Reference_Entity_State_PDU => Reference_PDU,
    Status                 => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
  end if;
  .
    end Filter_PDU_List;
```

Filter_List.Unit tc "Filter_List.Fire_Elevation.Filter_Orientation Unit"\l 4§

Procedure Syntax:

```
Filter_List.Fire_Elevation.Filter_Orientation(
  The_List                : in out DL_Linked_List_Types.
                                                    Fire_List.PTR;
  Threshold_1             : in     Numeric_Types.FLOAT_32_BIT;
  Threshold_2             : in     Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in     DIS_Types.
                                                    AN_ENTITY_STATE_PDU;
  Status                  :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Fire PDUs by elevation.

This is an instantiation of the generic package Generic_Filter_List_By_Orientation with the Orientation procedure instantiated to filter by elevation and a call to the package's generic procedure Filter_Orientation which returns a sublist of Fire PDUs that meet the specified elevation threshold.

Input/Output Parameters:

The_List - A pointer to a list of Fire PDUs which will be evaluated with respect to the Reference_Entity_State_PDU.

Input Parameters:

Threshold_1 - Specifies the first angle in degrees of the elevation threshold range.

Threshold_2 - Specifies the second angle in degrees of the elevation threshold range.

Reference_Entity_State_PDU - Describes the reference entity and its geocentric position in meters.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_EL_FIRE_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Filter_List,
    Global_Data,
    Numeric_Types;

procedure Filter_PDU_List is

   -- Declare local variables

   Call_Status    : DL_Status.STATUS_TYPES;

   El_Threshold_1 : Numeric_Types.FLOAT_32_BIT := 45.0
   El_Threshold_2 : Numeric_Types.FLOAT_32_BIT := 65.0

(Assume that a list of Fire PDUs and a reference Entity State PDU with the orientation and
location fields initialized are known);

   Fire_PDUs      : DL_Linked_List_Types.Fire_List.PTR
                                := Global_Data.Fire_PDU_List;

   Reference_PDU  : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                                := Global_Data.Reference_Position;
   .
begin

   .
   --Delete PUUs which do not meet the specified criteria.
   Filter_List.Fire_Elevation.Filter_Orientation(
     The_List                => Fire_PDUs,
     Threshold_1               => El_Threshold_1,
     Threshold_2               => EL_Threshold_2,
     Reference_Entity_State_PDU => Reference_PDU,
     Status                  => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
   end if;
   .
     end Filter_PDU_List;
```

Filter_List.Entity_State_Max_Velocity.Filter_Velocity Unit tc "Filter_List.Entity_State_Max_Velocity.Filter_Velocity Unit"\l 4§

Procedure Syntax:

```
Filter_List.Entity_State_Max_Velocity.Filter_Velocity(
  The_List  : in out DL_Linked_List_Types.Entity_State_List.PTR;
  Threshold : in     Numeric_Types.FLOAT_32_BIT;
  Status    :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Entity State PDUs by less than or equal to a maximum velocity.

This is an instantiation of the generic package Generic_Filter_List_By_Velocity with the Velocity procedure instantiated to filter by maximum velocity and a call to the package's generic procedure Filter_Velocity which returns a sublist of Entity State PDUs that meet the specified velocity threshold.

Input/Output Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to the threshold.

Input Parameters:

Threshold - Specifies the maximum velocity in meters per second for the entity.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_MAX_VEL_ENT_ST_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Filter_List,
    Global_Data,
    Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Entity State PDUs with velocity fields initialized is known);

  Entity_State_PDUs  : DL_Linked_List_Types.Entity_State_List.PTR
                                        := Global_Data.Entity_State_PDU_List;

  Maximum_Velocity  : Numeric_Types.FLOAT_32_BIT := 600.0

  .
  .

begin

  .
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Entity_State_Max_Velocity.Filter_Velocity(
    The_List          => Entity_State_PDUs,
    Threshold          => Maximum_Velocity,
    Status           => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Filter_PDU_List;
```

Filter_List.Entity_State_Min_Velocity.Filter_Velocity Unit tc "Filter_List.Entity_State_Min_Velocity.Filter_Velocity Unit"\l 4§

Procedure Syntax:

```
Filter_List.Entity_State_Min_Velocity.Filter_Velocity(
  The_List  : in out DL_Linked_List_Types.Entity_State_List.PTR;
  Threshold : in     Numeric_Types.FLOAT_32_BIT;
  Status    :    out DL_Status.STATUS_TYPE);
```

Remarks:

Filters a list of Entity State PDUs by greater than or equal to a minimum velocity.

This is an instantiation of the generic package Generic_Filter_List_By_Velocity with the Velocity procedure instantiated to filter by minimum velocity and a call to the package's generic procedure Filter_Velocity which returns a sublist of Entity State PDUs that meet the specified velocity threshold.

Input/Output Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to the threshold.

Input Parameters:

Threshold - Specifies the minimum velocity in meters per second for the entity.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.FILT_MIN_VEL_ENT_ST_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Filter_List,
    Global_Data,
    Numeric_Types;

procedure Filter_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Entity State PDUs with velocity fields initialized is known);

  Entity_State_PDUs  : DL_Linked_List_Types.Entity_State_List.PTR
                                        := Global_Data.Entity_State_PDU_List;

  Minimum_Velocity  : Numeric_Types.FLOAT_32_BIT := 200.0

  .
  .

begin

  .
  .
  --Delete PUUs which do not meet the specified criteria.
  Filter_List.Entity_State_Min_Velocity.Filter_Velocity(
    The_List          => Entity_State_PDUs,
    Reference_Position => Reference_Location,
    Threshold         => Minimum_Velocity,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Filter_PDU_List;
```

Hashing.Delete_Item Unit tc "Hashing.Delete_Item Unit"\l 4§

Procedure Syntax:

```
Hashing.Delete_Item(
  Entity_State_PDU : in     DIS_Types.AN_ENTITY_STATE_PDU;
  Status           :    out DL_Status.STATUS_TYPE);
```

Remarks:

Deletes the entity from the Hash Table.

(The Smooth_Position_Update.Smooth_Entity unit maintains a hash table of entity information for entities that are being dead-reckoned and smoothed. This table is unbounded and should have expired entity's data removed from it.)

Input Parameters:

Entity_State_PDU - Contains the information needed to delete the entity from the hash table.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.ITEM_NOT_FOUND - Indicates that the Entity State PDU data was not in the hash table.
- DL_Status.DELETE_FREE_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Status,
    DIS_Types,
    Global_Data,
    Hashing;

procedure Delete_Expired_Entity is

  -- Declare local variables

  Call_Status : DL_Status.STATUS_TYPES;

(Assume that the expired entity has been stored in a global variable)

  Delete_PDU : DIS_Types.AN_ENTITY_STATE_PDU
                              := Global_Data.Entity_State_PDU;

  .
  .

begin

  .
  .
  --Delete entity information from the hash table.
  Hashing.Delete_Item(
    Entity_State_PDU => Delete_Item,
    Status           => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Delete_Expired_Entity
```

Orientation_Conversions.Eulers_To_Local_Orientation Unit tc "Orientation_Conversions.Eulers_To_Local_Orientation Unit"\l 4§

Procedure Syntax:

```
Orientation_Conversions.Eulers_To_Local_Orientation(
  Euler_Angles       : in     DIS_Types.AN_EULER_ANGLES_RECORD;
  Geodetic_Coordinates : in   DL_Types.
                                              THE_GEODETIC_COORDINATES;
  Local_Orientation   :   out DL_Types.LOCAL_ORIENTATION;
  Status              :   out DL_Status.STATUS_TYPE);
```

Remarks:

Converts from Euler Angles to local roll, pitch, and heading.

Input Parameters:

Euler_Angles - Defines the entity's location in the geocentric coordinate system in terms of pitch (rotation about the lateral or X axis), roll (rotation about the longitudinal or Y axis), and yaw (rotation about the altitude or Z axis). Angles are defined in degrees.

Geodetic_Position - Defines the entity's position in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Output Parameters:

Local_Orientation - Defines the entity's orientation in terms of roll (degrees), pitch (degrees), and heading (degrees).

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.EUL_ORI_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Status,
   DL_Types,
   DIS_Types,
   Orientation_Conversions;

procedure Orientation_Conversion is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

  Euler_Orientation  : DIS_Types.AN_EULER_ANGLES_RECORD
                                      := (Psi   => 75.0,
                                          Theta => 50.0,
                                          Phi   => 45.0);

  Geodetic_Location : DL_Types.The_GEODETIC_COORDINATES
                                      := (Latitude  => 80.0
                                          Longitude => 145.0
                                          Altitude  => 6_500.0);

  Local_Orientation : DL_Types.LOCAL_ORIENTATION;

  .
  .

begin

  .
  .
  -- Convert from Euler Angles to local roll, pitch and heading.
  Orientation_Conversions.Eulers_To_Local_Orientation(
    Euler_Angles        => Euler_Orientation,
    Geodetic_Coordinates => Geodetic_Location,
    Local_Orientation    => Local_Orientation,
    Status              => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Orientation_Conversion;
```

Orientation_Conversions.Local_Orientation_To_Eulers Unit tc "Orientation_Conversions.Local_Orientation_To_Eulers Unit"\l 4§

Procedure Syntax:

```
Orientation_Conversions.Local_Orientation_To_Eulers(
  Geodetic_Coordinates : in     DL_Types.THE_GEODETIC_COORDINATES;
  Local_Orientation    : in     DL_Types.LOCAL_ORIENTATION;
  Euler_Angles         :    out DIS_Types.AN_EULER_ANGLES_RECORD;
  Status               :    out DL_Status.STATUS_TYPE);
```

Remarks:

Converts from local roll, pitch, and heading to Euler Angles.

Input Parameters:

Local_Orientation - Defines the entity's orientation in terms of roll (degrees), pitch (degrees), and heading (degrees).

Geodetic_Position - Defines the entity's position in terms of latitude (degrees), longitude (degrees), and altitude (meters).

Output Parameters:

Euler_Angles - Defines the entity's location in the geocentric coordinate system in terms of pitch (rotation about the lateral or X axis), roll (rotation about the longitudinal or Y axis), and yaw (rotation about the altitude or Z axis). Angles are defined in degrees.

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.ORI_EUL_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Status,
    DL_Types,
    DIS_Types,
   Orientation_Conversions;

procedure Orientation_Conversion is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

  Euler_Orientation  : DIS_Types.AN_EULER_ANGLES_RECORD;

  Geodetic_Location : DL_Types.The_GEODETIC_COORDINATES
                                      := (Latitude  => 80.0
                                          Longitude => 145.0
                                          Altitude  => 6_500.0);

  Local_Orientation : DL_Types.LOCAL_ORIENTATION
                                      := (Roll    => 75.0,
                                          Pitch   => 50.0,
                                          Heading => 45.0);

  .
  .

begin

  .
  .
  -- Convert from local roll, pitch and heading to Euler Angles.
  Orientation_Conversions.Local_To_Eulers_Orientation(
    Geodetic_Coordinates => Geodetic_Location,
    Local_Orientation    => Local_Orientation,
    Euler_Angles         => Euler_Orientation,
    Status               => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Orientation_Conversion;
```

Smooth_Position_Update.Smooth_Entity Unit tc
"Smooth_Position_Update.Smooth_Entity Unit"\l 4§

Procedure Syntax:

```
Smooth_Position_Update.Smooth_Entity(
  Entity_State_PDU       : in out DIS_Types.AN_ENTITY_STATE_PDU;
  Elapsed_Time           : in     INTEGER;
  Alpha_Beta_Coefficient  : in     DL_Math.PERCENT := 0.0;
  Rate_Limiter_Tolerance  : in     DL_Math.PERCENT := 0.0;
  Rate_Smoother_Tolerance : in     DL_Math.PERCENT := 0.0;
  Use_Alpha_Beta         : in     BOOLEAN := FALSE;
  Use_Rate_Limiter       : in     BOOLEAN := FALSE;
  Use_Rate_Smoother      : in     BOOLEAN := FALSE;
  Status                 :    out DL_Status.STATUS_TYPE) is
```

Remarks:

Dead-reckons the entity and provides filters for adjusting new Entity State PDU updates which conflict with the dead-reckoned data. At least one of the filters has to be used. If no filtering is wanted, the Dead_Reckoning.Update_Position unit should be called instead of this unit.

This unit calls the designated filter to either dead-reckon the input data or to compensates for time discrepancies and network anomalies by adjusting the data.

Note: All filters are written with the assumption that the entity is being dead-reckoned using the Smooth_Entity unit and this unit is called every timeslice. If no flags are set, a Status of DL_Status.NO_FLAGS_SET will be returned and the data will not be updated.

Input/Output Parameters:
Entity_State_PDU - Contains either the currently displayed or new Entity State PDU position/velocity update information.
Input Parameters:

Elapsed_Time - The time (in microseconds) that has elapsed since the last position update. *Alpha_Beta_Coefficient* - Weight used by the Alpha_Beta_Filter which is given to the difference between the new position and the last position dead-reckoned. This weight is defined as a percentage between 0.0..1.0. The weight given to the new position is 1.0 minus the Alpha_Beta_Coefficient. For example, if the coefficient value is 0.1 the weight of confidence in the new position would be 90 percent and the weight of confidence in the difference between the new position and the dead-reckoned position would be 10 percent. These two values would be added to get the dampen position.
Rate_Limiter_Tolerance - The tolerance that is used by the Rate_Limiter unit to smooth input data for screen display purposes when the input data exceeds the next predicted position. This tolerance is defined as a percentage of the difference

between the input position and the dead-reckoned position. The offset is dynamically calculated based on this percentage. The amount calculated is then added to the dead-reckoned position to get an adjusted position which avoids a big jump on the display screen. This percent is a value between 0.0 and 1.0. For example, if the positional difference is 30 meters and the tolerance is set at 0.1, then the new position will be limited to the dead-reckoned position + 3 meters.*Rate_Smoother_Tolerance* - The tolerance that is used by the Rate_Change_Smoother unit to smooth the input data for screen display purposes when the input data exceeds the next predicted position. This tolerance is defined as a percentage of the difference between the last position update and the dead-reckoned position which is used to dynamically calculate a variance for each position/velocity component value. If the difference between the input position/velocity and predicted position/velocity exceeds this variance, then an adjustment is made. For example, if the difference between the last position update and the next predicted update is 30 meters and the tolerance is set at 0.1, then a variance of 3 meters would be calculated and allowed without any adjustment. If the difference between the input position and the predicted position exceeds 3 meters, then the input data would be adjusted. The adjustment would be based on the difference between the input position and the predicted position and would be made over several timeslices.*Use_Alpha_Beta* - Flag which if set TRUE will cause the alpha-betasmoothing algorithm to be invoked.*Use_Rate_Smoother* - Flag which if set TRUE will cause the Rate_Change_Limiter smoothing algorithm to be invoked.*Use_Rate_Limiter* - Flag which if set TRUE will cause the Rate_Limiter smoothing algorithm to be invoked.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.NO_FLAGS_SET - indcates that the procedure was called with all the Use.. flags defaulted to FALSE.
- DL_Status.SMOOTH_ENTITY_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Math,
    DL_Status,
    DIS_Types,
    Global_Data,
    Smooth_Position_Update;

procedure Orientation_Conversion is

  -- Declare local variables

  Call_Status     : DL_Status.STATUS_TYPES;

(Assume that the following information is known: an Entity State PDU with the kinematic data
and dead-reckoning algorithm initialized, current time and time of last update.)

  Elapsed_Time    : INTEGER
                          := Global_Data.Current_Time -
                                Global_Data.Last_Update

  Entity_To_Smooth : DIS_Types.AN_ENTITY_STATE_PDU
                                := Global_Data.Entity_State_PDU;

Smooth_Data       : BOOLEAN := TRUE;

  Tolerance       : DL_Math.PERCENT
                                := 10.0;
  .

begin

  .
  -- Smooth the position update using the Rate_Change_Smoother
  -- algorithm.
  Smooth_Position_Update.Smooth_Entity(
    Entity_State_PDU       => Entity_To_Smooth,
    Rate_Smoother_Tolerance => Tolerance,
    Use_Rate_Smoother      => Smooth_Data,
    Status                 => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
  end if;
  .
    end Smooth_Position;
```

Sort_List.Detonation_Distance.Sort_Distance Unit tc "Sort_List.Detonation_Distance.Sort_Distance Unit"\l 4§

Procedure Syntax:

```
Sort_List.Detonation_Distance.Sort_Distance(
  The_List          : in out DL_Linked_List_Types.
                                                Detonation_List.PTR;
  Ascending         : in     BOOLEAN := TRUE;
  Reference_Position : in    DIS_TYPES.A_WORLD_COORDINATE;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Sorts a list of Detonation PDUs by the distance from the PDU location to a reference point.

This is an instantiation of the generic package Generic_Sort_List_By_Distance which instantiates Generic_Binary_Insertion_Sort to sort in either ascending or descending order according to the input parameter Ascending and a call to the package's generic procedure Sort_Distance which returns a list of Detonation PDUs sorted according to the distance from the input Reference_Position.

Note: The "Ascending" parameter is optional. If it is omitted, it defaults to ascending order which means that the first PDU in the list will be the one that is closest to the Reference_Position.

In/Out Parameters:

The_List - A pointer to a list of Detonation PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Used to calculate the distance from it to each PDU location in the list.

Ascending - The default is TRUE. If this is set to FALSE, the list will be sorted in descending order.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.

- DL_Status.SORT_DIST_DETON_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Global_Data,
    Sort_List;

procedure Sort_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Detonation PDUs with the location fields initialized is known);

  Detonation_PDUs    : DL_Linked_List_Types.Detonation_List.PTR
                                    := Global_Data.Detonation_PDU_List;

  Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                  := (X => 2_000.0,
                                      Y => 4_025.0,
                                      Z => 1_025.0);
  .

begin
  .
  .
  -- Sort in ascending order.
  Sort_List.Detonation_Distance.Sort_Distance(
    The_List          => Detonation_PDUs,
    Reference_Position => Reference_Location,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
    end Sort_PDU_List;
```

Sort_List.Entity_State_Distance.Sort_Distance Unit tc "Sort_List.Entity_State_Distance.Sort_Distance Unit"\l 4§

Procedure Syntax:

```
Sort_List.Entity_State_Distance.Sort_Distance(
  The_List          : in out DL_Linked_List_Types.
                                                Entity_State_List.PTR;
  Ascending         : in     BOOLEAN := TRUE;
  Reference_Position : in    DIS_TYPES.A_WORLD_COORDINATE;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Sorts a list of Entity State PDUs by the distance from the entity to a reference point.

This is an instantiation of the generic package Generic_Sort_List_By_Distance which instantiates Generic_Binary_Insertion_Sort to sort in either ascending or descending order according to the input parameter Ascending and a call to the package's generic procedure Sort_Distance which returns a list of Entity State PDUs sorted according to the distance from the input Reference_Position.

Note: The "Ascending" parameter is optional. If it is omitted, it defaults to ascending order which means that the first PDU in the list will be the one that is closest to the Reference_Position.

In/Out Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Used to calculate the distance from it to each entity in the list.

Ascending - The default is TRUE. If this is set to FALSE, the list will be sorted in descending order.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.

- DL_Status.SORT_DIST_ENT_ST_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Global_Data,
   Sort_List;

procedure Sort_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Entity State PDUs with the location fields initialized is known);

  Entity_State_PDUs  : DL_Linked_List_Types.
                                        Entity_State_List.PTR
                                         := Global_Data.
                                            Entity_State_PDU_List;

  Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                    := (X => 2_000.0,
                                        Y => 4_025.0,
                                        Z => 1_025.0);
  .
begin

  .
  .
  -- Sort in ascending order.
  Sort_List.Entity_State_Distance.Sort_Distance(
    The_List          => Entity_State_PDUs,
    Reference_Position => Reference_Location,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
  end if;
  .
    end Sort_PDU_List;
```

Sort_List.Fire_Distance.Sort_Distance Unit tc "Sort_List.Fire_Distance.Sort_Distance Unit"\l 4§

Procedure Syntax:

```
Sort_List.Fire_Distance.Sort_Distance(
  The_List          : in out DL_Linked_List_Types.Fire_List.PTR;
  Ascending         : in     BOOLEAN := TRUE;
  Reference_Position : in     DIS_TYPES.A_WORLD_COORDINATE;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Sorts a list of Fire PDUs by the distance from the PDU location to a reference point.

This is an instantiation of the generic package Generic_Sort_List_By_Distance which instantiates Generic_Binary_Insertion_Sort to sort in either ascending or descending order according to the input parameter Ascending and a call to the package's generic procedure Sort_Distance which returns a list of Fire PDUs sorted according to the distance from the input Reference_Position.

Note: The "Ascending" parameter is optional. If it is omitted, it defaults to ascending order which means that the first PDU in the list will be the one that is closest to the Reference_Position.

In/Out Parameters:

The_List - A pointer to a list of Fire PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Used to calculate the distance from it to each PDU location in the list.

Ascending - The default is TRUE. If this is set to FALSE, the list will be sorted in descending order.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.SORT_DIST_FIRE_FAILURE - Indicates an exception was

raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Global_Data,
   Sort_List;

procedure Sort_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Fire PDUs with the location fields initialized is known);

  Fire_PDUs         : DL_Linked_List_Types.Fire_List.PTR
                                    := Global_Data.Fire_PDU_List;

  Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                  := (X => 2_000.0,
                                      Y => 4_025.0,
                                      Z => 1_025.0);
  .

begin
  .
  .
  -- Sort in ascending order.
  Sort_List.Fire_Distance.Sort_Distance(
    The_List          => Fire_PDUs,
    Reference_Position => Reference_Location,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
    end Sort_PDU_List;
```

Sort_List.Laser_Distance.Sort_Distance Unit tc "Sort_List.Laser_Distance.Sort_Distance Unit"\l 4§

Procedure Syntax:

```
Sort_List.Laser_Distance.Sort_Distance(
  The_List          : in out DL_Linked_List_Types.Laser_List.PTR;
  Ascending         : in     BOOLEAN := TRUE;
  Reference_Position : in     DIS_TYPES.A_WORLD_COORDINATE;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Sorts a list of Laser PDUs by the distance from the PDU location to a reference point.

This is an instantiation of the generic package Generic_Sort_List_By_Distance which instantiates Generic_Binary_Insertion_Sort to sort in either ascending or descending order according to the input parameter Ascending and a call to the package's generic procedure Sort_Distance which returns a list of Laser PDUs sorted according to the distance from the input Reference_Position.

Note: The "Ascending" parameter is optional. If it is omitted, it defaults to ascending order which means that the first PDU in the list will be the one that is closest to the Reference_Position.

In/Out Parameters:

The_List - A pointer to a list of Laser PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Used to calculate the distance from it to each PDU location in the list.

Ascending - The default is TRUE. If this is set to FALSE, the list will be sorted in descending order.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.

- DL_Status.SORT_DIST_LASER_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_Types,
   Global_Data,
   Sort_List;

procedure Sort_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

(Assume that a list of Laser PDUs with the location fields initialized is known);

  Laser_PDUs        : DL_Linked_List_Types.Laser_List.PTR
                                        := Global_Data.Laser_PDU_List;

  Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                    := (X => 2_000.0,
                                        Y => 4_025.0,
                                        Z => 1_025.0);
  .

  .
  -- Sort in ascending order.
  Sort_List.Laser_Distance.Sort_Distance(
    The_List          => Laser_PDUs,
    Reference_Position => Reference_Location,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
    (call encountered an error - handle error)
    .
    .
  end if;
  .
  .
    end Sort_PDU_List;
```

Sort_List.Transmitter_Distance.Sort_Distance Unit tc "Sort_List.Transmitter_Distance.Sort_Distance Unit"\l 4§

Procedure Syntax:

```
Sort_List.Transmitter_Distance.Sort_Distance(
  The_List          : in out DL_Linked_List_Types.
                                                Transmitter_List.PTR;
  Ascending         : in     BOOLEAN := TRUE;
  Reference_Position : in    DIS_TYPES.A_WORLD_COORDINATE;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

Sorts a list of Transmitter PDUs by the distance from the PDU location to a reference point.

This is an instantiation of the generic package Generic_Sort_List_By_Distance which instantiates Generic_Binary_Insertion_Sort to sort in either ascending or descending order according to the input parameter Ascending and a call to the package's generic procedure Sort_Distance which returns a list of Transmitter PDUs sorted according to the distance from the input Reference_Position.

Note: The "Ascending" parameter is optional. If it is omitted, it defaults to ascending order which means that the first PDU in the list will be the one that is closest to the Reference_Position.

In/Out Parameters:

The_List - A pointer to a list of Transmitter PDUs which will be evaluated with respect to the Reference_Position.

Input Parameters:

Reference_Position - Used to calculate the distance from it to each PDU location in the list.

Ascending - The default is TRUE. If this is set to FALSE, the list will be sorted in descending order.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.

- DL_Status.SORT_DIST_TRANS_FAILURE - Indicates an exception was raised in this unit.

- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_Types,
    Global_Data,
    Sort_List;

procedure Sort_PDU_List is

  -- Declare local variables

  Call_Status       : DL_Status.STATUS_TYPES;

  Reference_Location : DIS_Types.A_WORLD_COORDINATE
                                  := (X => 2_000.0,
                                      Y => 4_025.0,
                                      Z => 1_025.0);

(Assume that a list of Transmitter PDUs with the location fields initialized is known);

  Transmitter_PDUs   : DL_Linked_List_Types.
                                      Transmitter_List.PTR
                                       := Global_Data.Transmitter_PDU_List;
  .

begin
  .
  -- Sort in ascending order.
  Sort_List.Transmitter_Distance.Sort_Distance(
    The_List          => Transmitter_PDUs,
    Reference_Position => Reference_Location,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
  end if;
  .
    end Sort_PDU_List;
```

Sort_List.Entity_State_Velocity.Sort_Velocity Unit tc "Sort_List.Entity_State_Velocity.Sort_Velocity Unit"\l 4§

Procedure Syntax:

```
Sort_List.Entity_State_Velocity.Sort_Velocity(
  The_List   : in out DL_Linked_List_Types.Entity_State_List.PTR;
  Descending : in     BOOLEAN := TRUE;
  Status     :    out DL_Status.STATUS_TYPE);
```

Remarks:

Sorts a list of Entity State PDUs according to the entity's velocity.

This is an instantiation of the generic package Generic_Sort_List_By_Velocity which instantiates Generic_Binary_Insertion_Sort to sort in either ascending or descending order according to the input parameter Descending and a call to the package's generic procedure Sort_Velocity which returns a list of Entity State PDUs sorted according to their velocity.

Note: The "Descending" parameter is optional. If it is omitted, it defaults to descending order which means that the first PDU in the list will be the one that has the greatest velocity.

In/Out Parameters:

The_List - A pointer to a list of Entity State PDUs which will be evaluated with respect to its calculated velocity magnitude.

Input Parameters:

Descending - The default is TRUE. If this is set to FALSE, the list will be sorted in ascending order.

Output Parameters:

Status - Indicates whether this unit encountered an error condition. One of the following status values will be returned:

- DL_Status.SUCCESS - Indicates the unit executed successfully.
- DL_Status.SORT_VEL_ENT_ST_FAILURE - Indicates an exception was raised in this unit.
- Other - If an error occurs in a call to a sub-routine, the procedure will terminate, and the status (error code) for the failed routine will be returned.

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    Global_Data,
    Sort_List;

procedure Sort_PDU_List is

  -- Declare local variables

  Call_Status      : DL_Status.STATUS_TYPES;

(Assume that a list of Entity State PDUs with the location fields initialized is known);

  Entity_State_PDUs : DL_Linked_List_Types.Entity_State_List.PTR
                                        := Global_Data.Entity_State_PDU_List;
  .
  .

begin

  .
  .
  -- Sort by velocity in descending order.
  Sort_List.Entity_State_Velocity.Sort_Velocity(
    The_List          => Entity_State_PDUs,
    Status            => Call_Status);

  if Call_Status /= DL_Status.SUCCESS then
     (call encountered an error - handle error)
     .
     .
  end if;
  .
  .
    end Sort_PDU_List;
```

Detailed Generic Package/Unit Descriptions:tc "Detailed Generic Package/Unit Descriptions"\l 3§

This section contains the detailed descriptions of each generic package which can be instantiated for different data types. Each generic package is listed by package name in alphabetical order. This section includes the data that is needed to instantiate the generic and an example of how the generic can be instantiated and called. The syntax for each callable unit in the generic package is included along with its purpose and the exceptions/status codes that can be raised/returned.

Generic_Binary_Insertion_Sort Package tc "Generic_Binary_Insertion_Sort Package"\l 4§

Remarks:

Sorts an array of items by inserting the current item to sort at the proper place within a list of previously sorted items. This package is instantiated by Generic_Sort_By_Distance and Generic_Sort_By_Velocity (if called as a stand alone unit, see exception block).

This package is instantiated with the following:

- ITEM to sort
- Array index type
- Array type (ITEMS)
- Less than function

Example:

```
with DL_Status,
    DIS_Types,
    Generic_Binary_Insertion_Sort,
    Global_Data;

procedure Sort_List(
  Status : out DL_Status.STATUS_TYPE) is

  -- Define a maximum size for the array structure to sort..
  K_Size : constant := 200;

  -- Define the structure of the item to sort.
  type SORT_RECORD is
    record
      PDU      : DIS_Types.AN_ENTITY_STATE_PDU;
      Distance : Numeric_Types.FLOAT_64_BIT;
  end record;

  -- Define the array structure to sort.
  type SORT_ARRAY is array (POSITIVE range <>) of SORT_RECORD;

  -- Define sort order function for generic instantiation.
  function Ascending is
    Left  : in SORT_RECORD;
    Right : in SORT_RECORD;
   return BOOLEAN;

-- Instantiate the generic sort routine to sort the Entity State PDU records in ascending order
according to the Distance value.
  package Sort_Ascending is new
    Generic_Binary_Insertion_Sort(
      ITEM = > SORT_RECORD,
      INDEX => POSITIVE,
      ITEMS => SORT_ARRAY,
      "<"   => Ascending);

  -- Declare local variables

  Index     : POSITIVE := POSITIVE'FIRST;

  PDU_Array  : SORT_ARRAY(Index..K_Size);
```

```
        -- Complete function definition.
        function Ascending is
          Left  : in SORT_RECORD;
          Right : in SORT_RECORD;
         return BOOLEAN is

        begin -- Ascending

          if Left.Distance < Right.Distance then
            return TRUE;
          else
            return FALSE;
         end if;

        end Ascending;

        .
        .

        begin -- Sort_List

      Status := DL_Status.SUCCESS;

      -- Initialize the PDU_Array with the data to sort from
      -- Global_Data.
      .
      .
      --Sort the PDU_Array
        Sort_Ascending.Sort(
          The_Items => PDU_Array)

        exception

  when others =>
    DL_Status := DL_Status.SORT_LIST_FAILURE;

end Sort_List;
```

The following unit can be called when this package is instantiated.

{Instantiation Package Name}.Sort Unit tc "{Instantiation Package Name}.Sort Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Sort(
  The_Items : in out ITEMS);
```

Remarks:

Sorts the list of items by putting the first two items in sorted order and then expanding the sort list by one. The new item is then placed at the proper position within the previously sorted list. This process is speeded up by dividing the sorted list into two parts and determining into which part the current item will be placed. This means that only half of a sublist must be searched in order to insert the item. This process is repeated until all the items are in sorted order.

Exceptions:

There are no exceptions raised from this unit; however, unknown Ada exceptions are not trapped in this unit. Any raised Ada exceptions are trapped in Generic_Sort_By_Velocity and Generic_Sort_By_Distance. If this unit is called independent of either of these units, an exception handler should be added to the calling procedure.

Example:

See package example.

Generic_Filter_List_By_Az_And_El Package tc "Generic_Filter_List_By_Az_And_El Package"\l 4§

Remarks:

Contains a unit that evaluates each PDU in a list and removes those PDUs which do not meet the specified azimuth and elevation thresholds.

This package is instantiated with the following:

- ITEM to Filter (i.e., pointer to a PDU record)
- Pointer to a linked list of ITEMs
- Function to return the appropriate status if an error occurs
- Function to return the location vector
- Procedure to delete a node in the linked list
- Function to find the position of an ITEM in the linked list
- Function to determine if the list is null
- Function to increment the pointer in the list
- Function to return the value of ITEM

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_PDU_Pointer_Types,
    DIS_Types,
    Generic_Filter_By_Az_And_El,
    Get_PDU_Data,
    Global_Data,
    Numeric_Types,
    Status_Functions;

procedure Filter_List(
  Status : out DL_Status.STATUS_TYPE) is

  -- Instantiate Generic_Filter_List_By_Az_And_El to filter a
  --list of Entity State PDU records by azimuth and elevation.
  package Filter_Entity_State_By_Az_And_El is new
   Generic_Filter_List_By_Az_And_El(
     ITEM        => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR,
     PTR         => DL_Linked_List_Types.Entity_State_List.PTR,
     Get_Status   => Status_Functions.
                                    Filt_Entity_State_Az_And_El_Status,
     PDU_Location => Get_PDU_Data.Entity_State_Location,
     Delete_Item_
     and_Free_
     Storage     => DL_Linked_List_Types.
                                    Entity_State_List_Utilities.
                                    Delete_Item_and_Free_Storage,
     Position_Of  => DL_Linked_List_Types.
                                    Entity_State_List_Utilities.Position_Of,
     Is_Null      => DL_Linked_List_Types.Entity_State_List.
                                    Is_Null,
     Tail_Of      => DL_Linked_List_Types.Entity_State_List.
                                    Tail_Of,
     Value_Of     => DL_Linked_List_Types.Entity_State_List.
                                    Value_Of);

  -- Declare local variables

  Call_Status      : DL_Status.STATUS_TYPES;

  Az_Threshold_1    : Numeric_Types.FLOAT_32_BIT := 45.0
  Az_Threshold_2    : Numeric_Types.FLOAT_32_BIT := 65.0
  El_Threshold_1    : Numeric_Types.FLOAT_32_BIT := 20.0
  El_Threshold_2    : Numeric_Types.FLOAT_32_BIT := 40.0;

(Assume that a list of Entity State PDUs and a reference Entity State PDU with the orientation
and location fields initialized are known);

  Entity_State_PDUs : DL_Linked_List_Types.Entity_State_List.PTR
                                    := Global_Data.Entity_State_PDU_List;

  Reference_PDU     : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                                    := Global_Data.Reference_Position;
```

```
   -- Define a local exception
   CALL_FAILURE : EXCEPTION;

begin -- Filter_List

   Status := DL_Status.SUCCESS;

   -- Delete all the items from the list which do not meet the
   -- azimuth and elevation threshold ranges.
   Filter_Entity_State_By_Az_And_El.Filter_Az_And_El(
     The_List                => Entity_State_PDUs,
     First_Az_Threshold      => Az_Threshold_1,
     Second_Az_Threshold     => Az_Threshold_2,
     First_El_Threshold      => El_Threshold_1,
     Second_El_Threshold     => El_Threshold_2,
     Reference_Entity_State_PDU => Reference_PDU,
     Status                  => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
     raise CALL_FAILURE;
   end if;

exception

   when CALL_FAILURE =>
    Status := Call_Status;
   when OTHERs =>
     Staus := DL_Status.FILT_AZ_AND_EL_FAILURE;

end Filter_List;
```

The following unit can be called when this package is instantiated.

{Instantiation Package Name}.Filter_Az_And_EL Unit tc "{Instantiation Package Name}.Filter_Az_And_El Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Filter_Az_And_El(
  The_List                : in out PTR;
  First_Az_Threshold      : in    Numeric_Types.FLOAT_32_BIT;
  Second_Az_Threshold     : in    Numeric_Types.FLOAT_32_BIT;
  First_El_Threshold      : in    Numeric_Types.FLOAT_32_BIT;
  Second_El_Threshold     : in    Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in    DIS_Types.
                                                  AN_ENTITY_STATE_PDU;
  Status                  :   out DL_Status.STATUS_TYPE);
```

Remarks:

This unit can be instantiated for any PDU type (see Filter_List for current instantiations and input/output definitions).

Example:

See package example.

Generic_Filter_List_By_Distance Package tc " Generic_Filter_List_By_Distance Package"§**Package**

Generic_Filter_List_By_Distance

Remarks:

Contains a unit that evaluates each PDU in a list and removes those PDUs which do not meet the specified Distance threshold.

This package is instantiated with the following:

- ITEM to Filter (i.e., pointer to a PDU record)
- Pointer to a linked list of ITEMs
- Function to return the appropriate status if an error occurs
- Function to return the location vector
- Procedure to delete a node in the linked list
- Function to find the position of an ITEM in the linked list
- Function to determine if the list is null
- Function to increment the pointer in the list
- Function to return the value of ITEM

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_PDU_Pointer_Types,
    DIS_Types,
    Generic_Filter_List_By_Distance,
    Get_PDU_Data,
    Global_Data,
    Numeric_Types,
    Status_Functions;

procedure Filter_List(
  Status : out DL_Status.STATUS_TYPE) is

  -- Instantiate Generic_Filter_List_By_Distasnce to filter a
  --list of Entity State PDU records by distance.
  package Filter_Entity_State_By_Distance is new
    Generic_Filter_List_By_Distance(
      ITEM        => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR,
      PTR         => DL_Linked_List_Types.Entity_State_List.PTR,
      Get_Status   => Status_Functions.
                                  Filt_Entity_State_Distance_Status,
      PDU_Location => Get_PDU_Data.Entity_State_Location,
      Delete_Item_
      and_Free_
      Storage     => DL_Linked_List_Types.
                                  Entity_State_List_Utilities.
                                  Delete_Item_and_Free_Storage,
      Position_Of  => DL_Linked_List_Types.
                                  Entity_State_List_Utilities.Position_Of,
      Is_Null      => DL_Linked_List_Types.Entity_State_List.
                                  Is_Null,
      Tail_Of      => DL_Linked_List_Types.Entity_State_List.
                                  Tail_Of,
      Value_Of     => DL_Linked_List_Types.Entity_State_List.
                                  Value_Of);

  -- Declare local variables

  Call_Status         : DL_Status.STATUS_TYPES;

(Assume that a list of Entity State PDUs with location fields initialized is known);

  Entity_State_PDUs    : DL_Linked_List_Types.
                                    Entity_State_List.PTR
                                        := Global_Data.
                                            Entity_State_PDU_List;

  Maximum_Distance    : Numeric_Types.FLOAT_64_BIT := 4_000.0
```

```
   Reference_Location   : DIS_Types.A_WORLD_COORDINATE
                                        := (X : 2_500.0,
                                            Y : 2_000.0,
                                            Z : 1_000.0);

   -- Define a local exception
   CALL_FAILURE : EXCEPTION;

begin -- Filter_List

   Status := DL_Status.SUCCESS;

   -- Delete all the items from the list which do not meet the
   -- distance threshold value.
   Filter_Entity_State_By_Distance.Filter_Distance(
     The_List           => Entity_State_PDUs,
     Reference_Position => Reference_Location,
     Threshold          => Maximum_Distance,
     Status             => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
      raise CALL_FAILURE;
   end if;

exception

   when CALL_FAILURE =>
    Status := Call_Status;
   when OTHERs =>
     Staus := DL_Status.FILT_DISTANCE_FAILURE;

end Filter_List;
```

The following unit can be called when this package is instantiated.

{Instantiation Package Name}.Filter_Distance Unit tc "{Instantiation Package Name}.Filter_Distance Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Filter_Distance(
  The_List           : in out PTR;
  Reference_Position : in     DIS_TYPES.A_WORLD_COORDINATE;
  Threshold          : in     Numeric_Types.FLOAT_64_BIT;
  Status             :    out DL_Status.STATUS_TYPE);
```

Remarks:

This unit can be instantiated for any PDU type (see Filter_List for current instantiations and input/output definitions).

Example:

See package example.

Generic_Filter_List_By_Orientation Package tc "Generic_Filter_List_By_Orientation Package"\l 4§

Remarks:

Evaluates each PDU in a list and removes those PDUs which do not meet the specified azimuth or elevation threshold. (Package can be instantiated to filter by either azimuth or elevation.)

This package is instantiated with the following:

- ITEM to Filter (i.e., pointer to a PDU record)
- Pointer to a linked list of ITEMs
- Procedure to filter by azimuth or elevation
- Function to return the appropriate status if an error occurs
- Function to return the location vector
- Procedure to delete a node in the linked list
- Function to find the position of an ITEM in the linked list
- Function to determine if the list is null
- Function to increment the pointer in the list
- Function to return the value of ITEM

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_PDU_Pointer_Types,
   DIS_Types,
   Filter,
   Generic_Filter_By_Orientation,
   Get_PDU_Data,
   Global_Data,
   Numeric_Types,
   Status_Functions;

procedure Filter_List(
  Status : out DL_Status.STATUS_TYPE) is

  -- Instantiate Generic_Filter_List_By_Orientation to filter a
  --list of Entity State PDU records by azimuth.
  package Filter_Entity_State_By_Azimuth is new
    Generic_Filter_List_By_Orientation(
      ITEM        => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR,
      PTR         => DL_Linked_List_Types.Entity_State_List.PTR,
      Orientation  => Filter.Azimuth,
      Get_Status   => Status_Functions.
                                    Filt_Entity_State_Az_Status,
      PDU_Location => Get_PDU_Data.Entity_State_Location,
      Delete_Item_
      and_Free_
      Storage     => DL_Linked_List_Types.
                                    Entity_State_List_Utilities.
                                    Delete_Item_and_Free_Storage,
      Position_Of  => DL_Linked_List_Types.
                                    Entity_State_List_Utilities.Position_Of,
      Is_Null     => DL_Linked_List_Types.Entity_State_List.
                                    Is_Null,
      Tail_Of     => DL_Linked_List_Types.Entity_State_List.
                                    Tail_Of,
      Value_Of    => DL_Linked_List_Types.Entity_State_List.
                                    Value_Of);
```

```
   -- Instantiate Generic_Filter_List_By_Orientation to filter a
   --list of Entity State PDU records by elevation.
   package Filter_Entity_State_By_Elevation is new
     Generic_Filter_List_By_Orientation(
       ITEM        => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR,
       PTR         => DL_Linked_List_Types.Entity_State_List.PTR,
       Orientation  => Filter.Elevation,
       Get_Status   => Status_Functions.
                                    Filt_Entity_State_El_Status,
       PDU_Location => Get_PDU_Data.Entity_State_Location,

       Delete_Item_and_Free_Storage => DL_Linked_List_Types.
                                    Entity_State_List_Utilities.
                                    Delete_Item_and_Free_Storage,

       Position_Of  => DL_Linked_List_Types.
                                    Entity_State_List_Utilities.Position_Of,
       Is_Null      => DL_Linked_List_Types.Entity_State_List.
                                    Is_Null,
       Tail_Of      => DL_Linked_List_Types.Entity_State_List.
                                    Tail_Of,
       Value_Of     => DL_Linked_List_Types.Entity_State_List.
                                    Value_Of);

   -- Declare local variables

   Call_Status    : DL_Status.STATUS_TYPES;

   Az_Threshold_1    : Numeric_Types.FLOAT_32_BIT := 45.0
   Az_Threshold_2    : Numeric_Types.FLOAT_32_BIT := 65.0
   El_Threshold_1    : Numeric_Types.FLOAT_32_BIT := 20.0
   El_Threshold_2    : Numeric_Types.FLOAT_32_BIT := 40.0;

(Assume that a list of Entity State PDUs and a reference Entity State PDU with the orientation
and location fields initialized are known);

   Azimuth_PDUs   : DL_Linked_List_Types.Entity_State_List.PTR
                                := Global_Data.Entity_State_PDU_List;

   Elevation_PDUs : DL_Linked_List_Types.Entity_State_List.PTR
                                := Global_Data.Entity_State_PDU_List;

   Reference_PDD  : DIS_Types.DIS_Types.AN_ENTITY_STATE_PDU
                                := Global_Data.Reference_Position;

   -- Define a local exception
   CALL_FAILURE : EXCEPTION;

begin -- Filter_List

   Status := DL_Status.SUCCESS;
```

```
   -- Delete all the items from the list which do not meet the
   -- azimuth threshold values.
   Filter_Entity_State_By_Azimuth.Filter_Orientation(
     The_List                => Azimuth_PDUs,
     Threshold_1             => Az_Threshold_1,
     Threshold_2             => Az_Threshold_2,
     Reference_Entity_State_PDU => Reference_PDU,
     Status                  => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
     raise CALL_FAILURE;
   end if;

   -- Delete all the items from the list which do not meet the
   -- Elevation threshold values.
   Filter_Entity_State_By_Elevation.Filter_Orientation(
     The_List                => Elevation_PDUs,
     Threshold_1             => El_Threshold_1,
     Threshold_2             => El_Threshold_2,
     Reference_Entity_State_PDU => Reference_PDU,
     Status                  => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
     raise CALL_FAILURE;
   end if;

exception

   when CALL_FAILURE =>
    Status := Call_Status;
   when OTHERs =>
     Staus := DL_Status.FILTER_AZ_AND_EL_FAILURE;

end Filter_PDU_List;
```

The following unit can be called when this package is instantiated.

{Instantiation Package Name}.Filter_Orientation Unit tc "{Instantiation Package Name}.Filter_Orientation Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Filter_Orientation(
  The_List                : in out PTR;
  Threshold_1             : in     Numeric_Types.FLOAT_32_BIT;
  Threshold_2             : in     Numeric_Types.FLOAT_32_BIT;
  Reference_Entity_State_PDU : in     DIS_Types.
                                                    AN_ENTITY_STATE_PDU;
  Status                  :    out DL_Status.STATUS_TYPE);
```

Remarks:

This unit can be instantiated for any PDU type and either azimuth or elevation orientation (see Filter_List for current instantiations and inputs/outputs).

Example:

See package example.

Generic_Filter_List_By_Velocity Package tc "Generic_Filter_List_By_Velocity Package"\l 4§

Remarks:

Evaluates each PDU in a list and remove those PDUs which do not meet the specified maximum or miminum velocity threshold. (Package can be instantiated to filter by either maximum or minimum velocity.)

This package is instantiated with the following:

- ITEM to Filter (i.e., pointer to a PDU record)
- Pointer to a linked list of ITEMs
- Procedure to filter by minimum or maximum velocity
- Function to return the appropriate status if an error occurs
- Function to return the velocity vector
- Procedure to delete a node in the linked list
- Function to find the position of an ITEM in the linked list
- Function to determine if the list is null
- Function to increment the pointer in the list
- Function to return the value of ITEM

Example:

```
with DL_Linked_List_Types,
   DL_Status,
   DIS_PDU_Pointer_Types,
   DIS_Types,
   Filter,
   Generic_Filter_By_Velocity,
   Get_PDU_Data,
   Global_Data,
   Numeric_Types,
   Status_Functions;

procedure Filter_List(
  Status : out DL_Status.STATUS_TYPE) is

  -- Instantiate Generic_Filter_List_By_Velocity to filter a
  --list of Entity State PDU records by maximum velocity.
  package Filter_Entity_State_By_Max_Velocity is new
    Generic_Filter_List_By_Velocity(
      ITEM        => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR,
      PTR         => DL_Linked_List_Types.Entity_State_List.PTR,
      Velocity    => Filter.Maximum_Velocity,
      Get_Status  => Status_Functions.
                                    Filt_Entity_State_Max_Vel_Status,
      PDU_Velocity => Get_PDU_Data.Entity_State_Velocity,
      Delete_Item_
      and_Free_
      Storage     => DL_Linked_List_Types.
                                    Entity_State_List_Utilities.
                                    Delete_Item_and_Free_Storage,
      Position_Of  => DL_Linked_List_Types.
                                     Entity_State_List_Utilities.Position_Of,
      Is_Null     => DL_Linked_List_Types.Entity_State_List.
                                    Is_Null,
      Tail_Of     => DL_Linked_List_Types.Entity_State_List.
                                     Tail_Of,
      Value_Of    => DL_Linked_List_Types.Entity_State_List.
                                    Value_Of);
```

```
-- Instantiate Generic_Filter_List_By_Velocity to filter a
--list of Entity State PDU records by minimum velocity.
package Filter_Entity_State_By_Min_Velocity is new
  Generic_Filter_List_By_Velocity(
    ITEM        => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR,
    PTR         => DL_Linked_List_Types.Entity_State_List.PTR,
    Velocity    => Filter.Minimum_Velocity,
    Get_Status  => Status_Functions.
                                   Filt_Entity_State_Min_Vel_Status,
    PDU_Velocity => Get_PDU_Data.Entity_State_Velocity,
    Delete_Item_
    and_Free_
    Storage     => DL_Linked_List_Types.
                                   Entity_State_List_Utilities.
                                   Delete_Item_and_Free_Storage,
    Position_Of => DL_Linked_List_Types.
                                   Entity_State_List_Utilities.Position_Of,
    Is_Null     => DL_Linked_List_Types.Entity_State_List.
                                   Is_Null,
    Tail_Of     => DL_Linked_List_Types.Entity_State_List.
                                   Tail_Of,
    Value_Of    => DL_Linked_List_Types.Entity_State_List.
                                   Value_Of);


-- Declare local variables

Call_Status     : DL_Status.STATUS_TYPES;
```

(Assume that a list of Entity State PDUs and a reference Entity State PDU with the orientation and location fields initialized are known);

```
Max_Velocitiy_PDUs : DL_Linked_List_Types.
                                   Entity_State_List.PTR
                                    := Global_Data.
                                        Entity_State_PDU_List;

Min_Velocity_PDUs : DL_Linked_List_Types.
                                   Entity_State_List.PTR
                                    := Global_Data.
                                        Entity_State_PDU_List;

Maximum_Velocity : Numeric_Types.FLOAT_32_BIT:= 600.0;
Minimum_Velocity : Numeric_Types.FLOAT_32_BIT:= 200.0;

-- Define a local exception
CALL_FAILURE : EXCEPTION;

begin -- Filter_List

Status := DL_Status.SUCCESS;
```

```
   -- Delete all the items from the list which do not meet the
   -- maximum velocity threshold.
   Filter_Entity_State_By_Max_Velocity.Filter_Velocity(
     The_List  => Max_Velocity_PDUs,
     Threshold => Maximum_Velocity;
     Status    => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
      raise CALL_FAILURE;
   end if;

   -- Delete all the items from the list which do not meet the
   -- minimum velocity threshold.
   Filter_Entity_State_By_Min_Velocity.Filter_Velocity(
     The_List  => Min_Velocity_PDUs,
     Threshold => Minimum_Velocity;
     Status    => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
      raise CALL_FAILURE;
   end if

exception

   when CALL_FAILURE =>
    Status := Call_Status;
   when OTHERs =>
     Staus := DL_Status.FILTER_VELOCITY_FAILURE;

end Filter_List;
```

The following unit can be called when this package is instantiated.

{Instantiation Package Name}.Filter_Velocity Unit tc "{Instantiation Package Name}.Filter_Velocity Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Filter_Velocity(
  The_List   : in out PTR;
  Threshold  : in     Numeric_Types.FLOAT_32_BIT;
  Status     :    out DL_Status.STATUS_TYPE);
```

Remarks:

This unit can be instantiated for any PDU type and either maximum or minimum velocity (see Filter_List for current instantiations and inputs/outputs).

Example:

See package example.

Generic_List Package tc "Generic_List Package"\l 4§

Remarks:

Contains primitive units to create and manage a generic double-linked, unbounded list of zero or more items in which items can be added to and removed from any position such that a strict linear ordering is maintained. The type of the item is immaterial to the behavior of the list which is an access-based structure. The ordering of items in the list is designated by the forward and backward linking of one item to the succeeding and previous items in the list. A null list contains zero items. If a list is not null, the first item is the head of the list. The sequence of items following a node is called the tail of the list. There can also be a preceding list to the referenced node.

This package is instantiated for the private type ITEM which is the item to store in the linked list and defines a private type PTR which accesses a node that contains a previous pointer, next pointer, and ITEM.

Currently this package is instantiated for the Detonation, Entity State, Fire, Laser, and Transmitter PDU data types in DL_Linked_List_Types package.

Example:

```
with DL_Status,
    DIS_PDU_Pointer_Types,
    Generic_List,
    Global_Data;

procedure Create_List(
  Status : out DL_Status.STATUS_TYPE) is

  -- Instantiate the list for a pointer to an Entity State PDU
  -- record.
  package Entity_State_List is new Generic_List(
   ITEM => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR);

  Init_PDU : DIS_Types.AN_ENTITY_STATE_PDU
         := Global_Data.Entity_State_Init_Data;

  New_PDU  : DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR

  The_List : Entity_State_List.PTR;

begin -- Create_List

  Status := DL_Status.SUCCESS;

  -- Create a list of 10 items initialized to the global record.
  for count in 1..10 loop
    New_PDU     := new DIS_Types.AN_ENTITY_STATE_PDU;
    New_PDU.All := Init_PDU;
    Entity_State_List.Construct_Top(
      The_Item => New_PDU
      Head     => The_List);
  end loop;

  -- Find the head of the list.
  while not Entity_State_List.Is_Null(
    Entity_State_List.Predecessor_Of(The_List)
  loop
   The_List := Entity_State_List.Predecessor_Of(The_List)
  end loop;
  .
  .
exception

  when OVERFLOW =>
   Status := DL_Status.OVERFLOW;
  when NOT_AT_HEAD =>
   Status := DL_Status.NOT_AT_HEAD;
  when OTHERS =>
   Status := DL_Status.CREATE_FAILURE;

end Create_List;
```

When this package is instantiated the following units are available for manipulation of the list: (These units will raise exceptions if errors are

encountered and should only be used in units which provide an exception handler to trap the error. See Generic_List_Utilities for the same units which return status codes instead of raising exceptions and other units which use the same data structure.)

{Instantiation Package Name}.Change_Item Unit tc "{Instantiation Package Name}.Change_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Change_Item(
  New_Item  : in ITEM;
  Null_Item : in ITEM;
  Old_Item  : in PTR)
```

Remarks:

Clears the current item and assigns the input item. (This unit is needed if the new item would not completely clear the existing item.)

Exceptions:

LIST_IS_NULL - raised if the input pointer is null;

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Change_The_Item for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Clear_List Unit tc "{Instantiation Package Name}.Clear_List Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Clear_List(
  The_List : in out PTR);
```

Remarks:

Sets the input pointer to null. (Does not deallocate memory, see Generic_List_Utilities.Free_List to deallocate memory for all the nodes in the list and set the pointer to null.)

Exceptions:

GEN_LIST_FAILURE - raised if any Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Clear_The_List for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Clear_Previous Unit tc "{Instantiation Package Name}.Clear_Previous Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Clear_Previous(
The_List : in PTR);

Remarks:

Sets a node's "previous" pointer to null.

Exceptions:

LIST_IS_NULL - raised if the input pointer is null.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Clear_Previous_Ptr for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Clear_Next Unit tc "{Instantiation Package Name}.Clear_Next Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Clear_Next(

The_List : in PTR);

Remarks:

Sets a node's "next" pointer to null.

Exceptions:

LIST_IS_NULL - raised if the input pointer is null;

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Clear_Next_Ptr for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Clear_Node Unit tc "{Instantiation Package Name}.Clear_Node Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Clear_Node(
The_List : in PTR);

Remarks:

Sets both "previous" and "next" pointers of a node to null. (Does not deallocate memory. See Free to deallocate the node's memory.)

Exceptions:

LIST_IS_NULL - raised if the input pointer is null.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Clear_The_Node for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Construct_Bottom Unit tc "{Instantiation Package Name}.Construct_Bottom Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Construct_Bottom(
  The_Item : in ITEM,
  Tail : in out PTR);

Remarks:

Adds a new node at the end of the list.

Exceptions:

OVERFLOW - raised if the list cannot grow large enough to hold the item.

NOT_AT_END - raised if the pointer does not point to the last item in the list.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Insert_Item_End for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Construct_Top Unit tc "{Instantiation Package Name}.Construct_Top Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Construct_Top(
  The_Item : in ITEM,
  Head : in out PTR);

Remarks:

Adds a new node at the head of the list.

Exceptions:

OVERFLOW - raised if the list cannot grow large enough to hold the item.

NOT_AT_HEAD - raised if the pointer does not point to the first item in the list.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Insert_Item_Top for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Copy Unit tc "{Instantiation Package Name}.Copy Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Copy(
  From_List : in     PTR;
  To_List   : in out PTR);
```

Remarks:

Copies the items from one list to another list.

Exceptions:

OVERFLOW - raised if the destination list cannot grow large enough to hold the source list.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Copy_List for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Free Unit tc "{Instantiation Package Name}.Free Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Free(
  The_Node : in out PTR);

Remarks:

Deallocates storage and sets the pointer to null.

Exceptions:

GEN_LIST_FAILURE - raised if any Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Free_Node for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Set_Head Unit tc "{Instantiation Package Name}.Set_Head Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Set_Head(
  Head   : in PTR;
  To_Item : in ITEM);

Remarks:

Assign the input item to the node pointed to by the input pointer.

Exceptions:

LIST_IS_NULL is raised if the input pointer is null.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Assign_Item for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Swap_Tail Unit tc "{Instantiation Package Name}.Swap_Tail Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Swap_Tail(
  List_Tail : in     PTR;
  The_List  : in out PTR);
```

Remarks:

Exchanges the tail of List_Tail for the tail of The_List . If List_Tail is pointing to null, the effect of this call would be to concatenate the two lists in List_Tail and set The_List to null.

Exceptions:

NOT_AT_HEAD - raised if The_List.Previous pointer is null.

LIST_IS_NULL - raised if The_List ponter is null.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Swap_Tails for the same unit with a status code returned.

Example:

See package example

{Instantiation Package Name}.Is_Equal Unit tc "{Instantiation Package Name}.Is_Equal Unit"\l 5§

Function Syntax:

```
{Instantiation Package Name}.Is_Equal(
  List_1 : in PTR;
  List_2 : in PTR)
 return BOOLEAN;
```

Remarks:

Return TRUE if the lists have the same state (i.e., equal number of items and the order and value of their items are the same).

Exceptions:

GEN_LIST_FAILURE - raised if any Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Check_List_Equal for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Is_Null Unit tc "{Instantiation Package Name}.Is_Null Unit"\l 5§

Function Syntax:

```
{Instantiation Package Name}.Is_Null(
  The_List : in PTR)
 return BOOLEAN;
```

Remarks:

Returns TRUE if there are no items in the list.

Exceptions:

GEN_LIST_FAILURE - raised if any Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Check_Null for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Length_Of Unit tc "{Instantiation Package Name}.Length_Of Unit"\l 5§

Function Syntax:

```
{Instantiation Package Name}.Length_Of(
  The_List : in PTR)
 return NATURAL;
```

Remarks:

Returns the number of items in the list (traverses from input point to bottom).

Exceptions:

GEN_LIST_FAILURE - raised if any Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Get_Size for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Predecessor_Of Unit tc "{Instantiation Package Name}.Predecessor_Of Unit"\l 5§

Function Syntax:

```
{Instantiation Package Name}.Predecessor_Of(
  The_List : in PTR)
return PTR;
```

Remarks:

Returns a list containing the sequence of nodes preceding the node pointed to by the input pointer.

Exceptions:

LIST_IS_NULL - raised if the input pointer is null.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Get_Previous for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Tail_Of Unit tc "{Instantiation Package Name}.Tail_Of Unit"\l 5§

Function Syntax:

```
{Instantiation Package Name}.Tail_Of(
  The_List : in PTR)
return PTR;
```

Remarks:

Returns a list containing the sequence of nodes after the node pointed to by the

input pointer.

Exceptions:

LIST_IS_NULL - raised if the input pointer is null.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Get_Next for the same unit with a status code returned.

Example:

See package example.

{Instantiation Package Name}.Value_Of Unit tc "{Instantiation Package Name}.Value_Of Unit"\l 5§

Function:

```
{Instantiation Package Name}.Value_Of(
  The_List : in PTR)
 return ITEM;
```

Remarks:

Returns the value of the item at the head of the list.

Exceptions:

LIST_IS_NULL - raised if the input pointer is null.

GEN_LIST_FAILURE - raised if any other Ada exception is raised in the unit.

Note: See Generic_List_Utilities.Get_Previous for the same unit with a status code returned.

Example:

See package example.

Generic_List_Utilities Package tc " Generic_List_Utilities Package"§

Remarks:

Contains units to manipulate a linked list instantiation of Generic_List.

This package is instantiated with the instantiated generic list and a "less than" function.

Currently this package is instantiated for the Detonation, Entity State, Fire, Laser, and Transmitter PDU data types in DL_Linked_List_Types package.

Example:

```
with DL_Status,
    DIS_PDU_Pointer_Types,
    Generic_List,
    Generic_List_Utilities,
    Global_Data;

procedure Create_List(
  Status : out DL_Status.STATUS_TYPE) is

  package Entity_State_List is new Generic_List(
   ITEM => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR);

  -- Define function for instantiation of Generic_List_Utilities
  function Entity_State_Less_Than(
    Item_1 : in DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR;
    Item_2 : in DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR)
  return BOOLEAN;

  package Entity_State_List_Utilities is new
   Generic_List_Utilities(
    ITEM          => DIS_PDU_Pointer_Types.
                                        ENTITY_STATE_PDU_PTR,
    PTR           => Entity_State_List.PTR,
    Change_Item   => Entity_State_List.Change_Item,
    Clear_List    => Entity_State_List.Clear_List,
    Clear_Previous => Entity_State_List.Clear_Previous,
    Clear_Next    => Entity_State_List.Clear_Next,
    Clear_Node    => Entity_State_List.Clear_Node,
    Construct_Bottom => Entity_State_List.Construct_Bottom,
    Construct_Top => Entity_State_List.Construct_Top,
    Copy          => Entity_State_List.Copy,
    Free          => Entity_State_List.Free,
    Set_Head      => Entity_State_List.Set_Head,
    Swap_Tail     => Entity_State_List.Swap_Tail,
    Is_Equal      => Entity_State_List.Is_Equal,
    Is_Null       => Entity_State_List.Is_Null,
    Length_Of     => Entity_State_List.Length_Of,
    Predecessor_Of => Entity_State_List.Predecessor_Of,
    Tail_Of       => Entity_State_List.Tail_Of,
    Value_Of      => Entity_State_List.Value_Of,
    "<"           => Entity_State_Less_Than);

New_PDU  : DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR
Init_PDDU : DIS_Types.AN_ENTITY_STATE_PDU
          := Global_Data.Entity_State_Init_Data;

The_List : Entity_State_List.PTR;

-- Define a local exception
CALL_FAILURE : EXCEPTION;

function Entity_State_Less_Than(
  Item_1 : in DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR;
```

```
      Item_2 : in DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR)
    return BOOLEAN is

    begin

      if Item_1.Location.X < Item_2.Location.X then
        return TRUE;
      else
        return FALSE;
      end if;

    end Entity_State_Less_Than

      begin -- Create_List

    Status := DL_Status.SUCCESS;

    -- Create a list of 10 items initialized to the global record.
    for count in 1..10 loop

      New_PDU     := new DIS_Types.AN_ENTITY_STATE_PDU;
      New_PDU.All := PDU_Init_Data;

      Entity_State_List_Utilities.Insert_Item_Top(
        The_Item => PDU
        The_List => The_List
        Status   => Call_Status);

      if Call_Status /= DL_Status.SUCCESS then
        raise CALL_FAILURE;
      end if

    end loop;

    -- Find the head of the list.
    Get_First_Item(
      The_List => The_List,
      The_Item => The_List,
      Status   => Call_Status);

      if Call_Status /= DL_Status.SUCCESS then
        raise CALL_FAILURE;
      end if
    .
      exception

  when CALL_FAILURE =>
    Status := Call_Status;
  when OTHERS =>
    Status := DL_Status.CREATE_FAILURE;

end Create_List;
```

When this package is instantiated the following units are available for manipulation of the list: (All of these units return status codes.)

{Instantiation Package Name}.Append_List Unit tc "{Instantiation Package Name}.Append_List Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Append_List(
  List_1 : in     PTR
  List_2 : in     PTR;
  Status :    out DL_Status.STATUS_TYPE);
```

Remarks:

Concatenates two lists. Append List_2 to List_1 and sets List_2 to null.

Status:

Success - DL_Status.SUCCESS;
Failure - DL_Status.APPEND_LIST_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Assign_Item Unit tc "{Instantiation Package Name}.Assign_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Assign_Item(
  Head    : in     PTR;
  To_Item : in     ITEM;
  Status  :    out DL_Status.STATUS_TYPE);
```

Remarks:

Assign the input item to the ITEM component of the node pointed to by the input pointer.

Status:

Success - DL_Status.SUCCESS;
Failure - DL_Status.ASSIGN_ITEM_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Change_The_Item Unit tc "{Instantiation Package Name}.Change_The_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Change_The_Item(
  New_Item  : in     ITEM;
  Null_Item : in     ITEM;
  Old_Item  : in     PTR;
  Status    :    out DL_Status.STATUS_TYPE)
```

Remarks:

Clears the current item and assigns the input item.

Status:

Success - DL_Status.SUCCESS;
Failure - DL_Status.CHANGE_ITEM_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Check_At_End Unit tc "{Instantiation Package Name}.Check_At_End Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Check_At_End(
  The_List : in     PTR;
  The_End  :    out BOOLEAN;
  Status   :    out DL_Status.STATUS_TYPE)
```

Remarks:

Sets The_End to TRUE if the item is the last item in the list; otherwise, The_End is set to FALSE;

Status:

Success - DL_Status.SUCCESS;
Failure - DL_Status.CHECK_AT_END_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Check_At_Head Unit tc "{Instantiation Package Name}.Check_At_Head Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Check_At_Head(
  The_List : in     PTR;
  The_Head :    out BOOLEAN;
  Status   :    out DL_Status.STATUS_TYPE)
```

Remarks:

Sets The_Head to TRUE if the item is the first item in the list; otherwise, The_Head is set to FALSE;

Status:

Success - DL_Status.SUCCESS;
Failure - DL_Status.CHECK_AT_HEAD_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Check_List_Equal Unit tc "{Instantiation Package Name}.Check_List_Equal Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Check_List_Equal(
List_1 : in PTR;
List_2 : in PTR;
Equal : out BOOLEAN;
Status : out DL_Status.STATUS_TYPE)

Remarks:

Sets Equal to TRUE if the lists have the same state (i.e., equal number of items and the order and value of their items are the same); otherwise, it is set to FALSE.

Status:

Success - DL_Status.SUCCESS;
Failure - DL_Status.CHECK_LIST_EQUAL_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Check_Null Unit tc "{Instantiation Package Name}.Check_Null Unit"\l 5§

Procedure Syntax:

*{Instantiation Package Name}.*Check_Null(
The_List : in PTR;
Null_Pointer : out BOOLEAN;
Status : out DL_Status.STATUS_TYPE)

Remarks:

Null_Pointer is set TRUE if there are no items in the list; otherwise, it is set to FALSE;.

Status:

Success - DL_Status.SUCCESS;
Failure - DL_Status.CHECK_NULL_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Clear_Previous_Ptr Unit tc "{Instantiation Package Name}.Clear_Previous_Ptr Unit"\l 5§

Procedure Syntax:

{Instantiation Package Name}.Clear_Previous_Ptr(
  The_List : in PTR;
  Status : out DL_Status.STATUS_TYPE);

Remarks:

Sets the node's previous pointer to null.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.CLEAR_PREVIOUS_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Clear_Next_Ptr Unit tc "{Instantiation Package Name}.Clear_Next_Ptr Unit"\l 5§

Procedure Syntax:

{Instantiation Package Name}.Clear_Next_Ptr(
  The_List : in PTR;
  Status : out DL_Status.STATUS_TYPE);

Remarks:

Sets the node's next pointer to null.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.CLEAR_NEXT_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Clear_The_List Unit tc "{Instantiation Package Name}.Clear_The_List Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Clear_The_List(
  The_List : in out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Sets the input pointer to null. (Does not deallocate memory, see Free_List to deallocate memory.)

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.CLEAR_LIST_FAILURE.

Example:

See package example.
{Instantiation Package Name}.Clear_The_Node Unit tc "{Instantiation Package Name}.Clear_The_Node Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Clear_The_Node(
  The_List : in     PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Sets both "previous" and "next" pointers of a node to null. (Does not deallocate the memory. See Free_Node to deallocate the memory.)

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.CLEAR_NODE_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Copy_List Unit tc "{Instantiation Package Name}.Copy_List Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Copy_List(
  From_List : in     PTR;
  To_List   : in out PTR;
  Status    :    out DL_Status.STATUS_TYPE);
```

Remarks:

Copy the items from one list to another list.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.COPY_LIST_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Delete_Item Unit tc "{Instantiation Package Name}.Delete_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Delete_Item(
  The_List     : in     PTR;
  At_Position  :        POSITIVE := 1;
  Deleted_Item :        PTR;
  Status       :    out DL_Status.STATUS_TYPE);
```

Remarks:

Removes an item from a list and returns the pointer to the deleted item. If no position value is input, it will remove the item at the head of the list. If a position is input, it will remove the item at the input position.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.DELETE_ITEM_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Delete_Item_And_Free_Storage Unit tc "{Instantiation Package Name}.Delete_Item_And_Free_Storage Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Delete_Item_And_Free_Storage(
  The_List     : in     PTR;
  At_Position  :        POSITIVE := 1;
  Status       :    out DL_Status.STATUS_TYPE);
```

Remarks:

Removes an item from a list and deletes the item's storage. If no position value is input, it will remove the item at the head of the list. If a position is input, it will remove the item at the input position.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.DELETE_FREE_ITEM_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Find_Positionn_Of_Item Unit tc "{Instantiation Package Name}.Find_Position_Of_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Find_Position_Of_Item(
  The_Item : in     ITEM;
  The_List : in     PTR;
  Position :    out POSITIVE;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Position is set to the numeric position of the item within the list.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.FIND_POSITION_OF_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Free_Node Unit tc "{Instantiation Package Name}.Free_Node Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Free_Node(
  The_Node : in out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Deallocates storage and sets the pointer to null.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.FREE_NODE_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Free_List Unit tc "{Instantiation Package Name}.Free_List Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Free_List(
  The_List : in out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Deletes each item from the list and clears the storage for that node. When all the nodes are deleted, the list is set to null.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.FREE_LIST_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Get_First_Item Unit tc "{Instantiation Package Name}.Get_First_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Get_First_Item(
  The_List : in     PTR;
  The_Item :    out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Returns a pointer to the item at the head of the list (i.e., traverses the list backward).

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.GET_FIRST_ITEM_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Get_Last_Item Unit tc "{Instantiation Package Name}.Get_Last_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Get_Last_Item(
  The_List : in     PTR;
  The_Item :    out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Returns a pointer to the last item in the list (i.e., traverses the list forward).

Status:

Success - DL_Status.SUCCESS.

Failure - DL_Status.GET_LAST_ITEM_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Get_Item Unit tc "{Instantiation Package Name}.Get_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Get_Item(
  The_List : in     PTR;
  The_Item :    out ITEM;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Returns the value of the item in the node pointed to by the input pointer.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.GET_ITEM_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Get_Previous Unit tc "{Instantiation Package Name}.Get_Previous Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Get_Previous(
  The_List : in     PTR;
  Previous :    out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Returns a list containing the sequence of nodes preceding the node pointed to by the input pointer.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.GET_PREVIOUS_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Get_Next Unit tc "{Instantiation Package Name}.Get_Next Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Get_Next(
  The_List : in     PTR;
  Next     :    out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Returns a list containing the sequence of nodes after the node pointed to by the input pointer.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.GET_NEXT_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Get_Sublist Unit tc "{Instantiation Package Name}.Get_Sublist Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Get_Sublist(
  The_Item : in     ITEM;
  The_List : in     PTR;
  Sublist  :    out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Returns a sublist whose head is the item given.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.GET_SUBLIST_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Get_Sublist Unit tc "{Instantiation Package Name}.Get_Sublist Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Get_Sublist(
  Position : in     POSITIVE;
  The_List : in     PTR;
  Sublist  :    out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Returns a sublist whose head is the item at the input position.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.GET_SUBLIST_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Get_Size Unit tc "{Instantiation Package Name}.Get_Size Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Get_Size(
  Position : in     POSITIVE;
  The_List : in     PTR;
  Sublist  :    out PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Returns the number of items in the list (i.e., traverses from point input to bottom).

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.GET_SIZE_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Insert_Item Unit tc "{Instantiation Package Name}.Insert_Item Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Insert_Item(
  The_Item       : in     ITEM;
  Into_List      : in     PTR;
  After_Position : in     POSITIVE;
  Status         :    out DL_Status.STATUS_TYPE);
```

Remarks:

Inserts an item into a list after a given position.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.INSERT_ITEM_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Insert_Item_End Unit tc "{Instantiation Package Name}.Insert_Item_End Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Insert_Item_End(
  The_Item : in     ITEM;
  The_List : in     PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Adds an item to the end of the list.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.INSERT_END_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Insert_Item_Top Unit tc "{Instantiation Package Name}.Insert_Item_Top Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Insert_Item_Top(
  The_Item : in     ITEM;
  The_List : in     PTR;
  Status   :    out DL_Status.STATUS_TYPE);
```

Remarks:

Adds an item at the top of the list.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.INSERT_TOP_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Insert_List Unit tc "{Instantiation Package Name}.Insert_List Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Insert_List(
  The_List      : in     PTR;
  Insert_List   : in     PTR;
  After_Position : in    POSITIVE;
  Status        :    out DL_Status.STATUS_TYPE);
```

Remarks:

Inserts a list into the list after the given position.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.INSERT_LIST_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Split Unit tc "{Instantiation Package Name}.Split Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Split(
  The_List    : in    PTR;
  At_Position : in    POSITIVE;
  Sublist     :    out PTR;
  Status      :    out DL_Status.STATUS_TYPE);
```

Remarks:

Breaks a list into two lists at a given position.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.SPLIT_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Straight_Insertion_Sort Unit tc "{Instantiation Package Name}.Straight_Insertion_Sort Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Straight_Insertion_Sort(
  The_List    : in out PTR;
  Status      :    out DL_Status.STATUS_TYPE);
```

Remarks:

Examines one item at a time and inserts it in the proper order relative to all previously processed items. Equal key values maintain their order.

Utilizes the imported function "<" to implement the sort.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.SORT_INSERT_FAILURE.

Example:

See package example.

{Instantiation Package Name}.Swap_Tails Unit tc "{Instantiation Package Name}.Swap_Tails Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Swap_Tails(
  List_Tail : in     PTR;
  The_List  : in out PTR;
  Status    :    out DL_Status.STATUS_TYPE);
```

Remarks:

Exchanges the tail of List_Tail for the tail of The_List. If List_Tail is pointing to null, the effect of this call would be to concatenate the two lists in List_Tail and set The_List to null.

Status:

Success - DL_Status.SUCCESS.
Failure - DL_Status.SWAP_TAILS_FAILURE.

Example:

See package example.

Generic_Sort_List_By_Distance Package tc
"Generic_Sort_List_By_Distance Package"\l 4§

Remarks:

Generic_Sort_List_By_Distance Package is a generic package which instantiates the Generic_Binary_Insertion_Sort for distance. This package can be instantiated for any PDU type (see Sort_List for current instantiations and input/output definitions).

This package is instantiated with the following:

- ITEM to Sort (i.e., pointer to a PDU record)
- Pointer to a linked list of ITEMs
- Function to change the ITEM in the node
- Function to return the lenght of the list
- Function to determine if the list is null
- Function to increment the pointer in the list
- Function to return the value of ITEM
- Function to return the appropriate status if an error occurs
- Function to return the location vector
- Function to return a null pointer

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_PDU_Pointer_Types,
    DIS_Types,
    Generic_Sort_By_Distance,
    Get_PDU_Data,
    Global_Data,
    Numeric_Types,
    Status_Functions;

procedure Sort_List(
  Status : out DL_Status.STATUS_TYPE) is

  -- Instantiate Generic_Sort_List_By_Distance to sort a
  -- list of Entity State PDU records by distance from the PDU
  -- location to the reference Entity State PDU.
  package Sort_Entity_State_By_Distance is new
    Generic_Sort_List_By_Distance(
      ITEM         => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR,
      PTR          => DL_Linked_List_Types.Entity_State_List.PTR,
      Change_Item  => DL_Linked_List_Types.
                                    Entity_State_List.Change_Item,
      Length_Of    => DL_Linked_List_Types.
                                    Entity_State_List.Length_Of,
      Is_Null      => DL_Linked_List_Types.Entity_State_List.
                                    Is_Null,
      Tail_Of      => DL_Linked_List_Types.Entity_State_List.
                                 Tail_Of,
      Value_Of     => DL_Linked_List_Types.Entity_State_List.
                                    Value_Of,
      Get_Status   => Status_Functions.
                                    Sort_Entity_State_Distance_Status,
      PDU_Location => Get_PDU_Data.Entity_State_Location,
      Null_Item    => Get_PDU_Data.
                                    Null_Entity_State_PDU_Pointer);

  -- Declare local variables

  Call_Status         : DL_Status.STATUS_TYPES;

(Assume that a list of Entity State PDUs and a reference Entity State PDU with the location fields
initialized are known);

  Entity_State_PDUs    : DL_Linked_List_Types.
                                     Entity_State_List.PTR
                                      := Global_Data.
                                          Entity_State_PDU_List;
```

```
   Reference_Location   : DIS_Types.A_WORLD_COORDINATE
                                        := (X : 2_500.0,
                                            Y : 2_000.0,
                                            Z : 1_000.0);

   -- Define a local exception
   CALL_FAILURE : EXCEPTION;

begin -- Sort_List

   Status := DL_Status.SUCCESS;

   -- Sort the items in the list in ascending order according to
   --the distance from the PDU location to the reference point.
   Sort_Entity_State_By_Distance.Sort_Distance(
     The_List          => Entity_State_PDUs,
     Reference_Position => Reference_Location,
     Status            => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
      raise CALL_FAILURE;
   end if;

exception

   when CALL_FAILURE =>
    Status := Call_Status;

   when OTHERs =>
     Staus := DL_Status.SORT_DISTANCE_FAILURE;

end Sort_List;
```

The following unit can be called when this package is instantiated.

{Instantiation Package Name}.Sort_Distance Unit tc "{Instantiation Package Name}.Sort_Distance Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Sort_Distance(
  The_List          : in out PTR;
  Ascending         : in     BOOLEAN := TRUE;
  Reference_Position : in    DIS_Types.A_WORLD_COORDINATE;
  Status            :    out DL_Status.STATUS_TYPE);
```

Remarks:

This unit can be instantiated for any PDU type (see Sort_List for current instantiations and inputs/outputs).

Example:

See package example.

Generic_Sort_List_By_Velocity Package tc "Generic_Sort_List_By_Velocity Package"\l 4§

Remarks:

Generic_Sort_List_By_Velocity Package is a generic package which instantiates the Generic_Binary_Insertion_Sort for velocity. This package can be instantiated for any PDU type (see Sort_List for current instantiations and input/output definitions).

This package is instantiated with the following:

- ITEM to Sort (i.e., pointer to a PDU record)
- Pointer to a linked list of ITEMs
- Function to change the ITEM in the node
- Function to return the lenght of the list
- Function to determine if the list is null
- Function to increment the pointer in the list
- Function to return the value of ITEM
- Function to return the appropriate status if an error occurs
- Function to return the velocity vector
- Function to return a null pointer

Example:

```
with DL_Linked_List_Types,
    DL_Status,
    DIS_PDU_Pointer_Types,
    DIS_Types,
    Generic_Sort_By_Velocity,
    Get_PDU_Data,
    Global_Data,
    Numeric_Types,
    Status_Functions;

procedure Filter_List(
  Status : out DL_Status.STATUS_TYPE) is

  -- Instantiate Generic_Sort_List_By_Velocity to sort a
  --list of Entity State PDU records by velocity.
  package Sort_Entity_State_By_Velocity is new
   Generic_Sort_List_By_Velocity(
     ITEM        => DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR,
     PTR         => DL_Linked_List_Types.Entity_State_List.PTR,
     Change_Item  => DL_Linked_List_Types.
                                    Entity_State_List.Change_Item,
     Length_Of    => DL_Linked_List_Types.
                                    Entity_State_List.Length_Of,
     Is_Null      => DL_Linked_List_Types.Entity_State_List.
                                    Is_Null,
     Tail_Of      => DL_Linked_List_Types.Entity_State_List.
                                    Tail_Of,
     Value_Of     => DL_Linked_List_Types.Entity_State_List.
                                    Value_Of,
     Get_Status   => Status_Functions.
                                    Sort_Entity_State_Velocity_Status,
     PDU_Velocity => Get_PDU_Data.Entity_State_Velocity,
     Null_Item    => Get_PDU_Data.
                                    Null_Entity_State_PDU_Pointer);

  -- Declare local variables

  Call_Status      : DL_Status.STATUS_TYPES;

(Assume that a list of Entity State PDUs and a reference Entity State PDU with the orientation
and location fields initialized are known);

  Entity_State_PDUs : DL_Linked_List_Types.
                                    Entity_State_List.PTR
                                     := Global_Data.
                                         Entity_State_PDU_List;

  -- Define a local exception
  CALL_FAILURE : EXCEPTION;

begin -- Filter_List

  Status := DL_Status.SUCCESS;
```

```
   -- Sort the list by velocity in descending order.
   Sort_Entity_State_By_Velocity.Sort_Velocity(
     The_List  => Entity_State_PDUs,
     Status    => Call_Status);

   if Call_Status /= DL_Status.SUCCESS then
      raise CALL_FAILURE;
   end if;

exception

   when CALL_FAILURE =>
    Status := Call_Status;

   when OTHERs =>
     Staus := DL_Status.SORT_VELOCITY_FAILURE;

end Filter_List;
```

The following unit can be called when this package is instantiated.

{Instantiation Package Name}.Sort_Velocity Unit tc "{Instantiation Package Name}.Sort_Distance Unit"\l 5§

Procedure Syntax:

```
{Instantiation Package Name}.Sort_Velocity(
   The_List   : in out PTR;
   Descending : in     BOOLEAN := TRUE;
   Status     :    out DL_Status.STATUS_TYPE);
```

Remarks:

This unit can be instantiated for any PDU type (see Sort_List for current instantiations and inputs/outputs).

Example:

See package example.

Chapter 3: How do you modify the DL? tc "Chapter 3: How do you modify the DL?"\l 1§

To expand the generic sort and filter routines to include other PDU types, execute the following steps:

1. Add the PDU record structure to DIS_Types Package Specification.

2. Add an access type for the record structure to DIS_PDU_Pointer_Types Package Specification.

3. Add Status value to DL_Status Package Specification.

4. Add a function to return the status to Status_Functions Package Specification and Body. (Use current functions as an example for adding new functions.)

5. Add a function to return the location/velocity to the Get_Data_PDU Package Specification and Body. (Use current functions as an example for adding new functions.)

6. Add the instantiation for the linked list to DL_Linked_List_Types Package Specification using the current instantiations in this package as an example (or see the Generic_List and Generic_List_Utilities Packages under Callable Routines).

7. Add the instantiation to the appropriate Sort_List or Filter_List Package Specification using the current instantiations in these packages as an example (or see the examples under Generic_Filter_List_By.... Packages in the Callable Routines section of this document).

Chapter 4: How do you troubleshoot the DL? tc "How do you troubleshoot the DL?"\l 1§

DL units return status codes which should be checked by the calling unit to determine whether an error occured. If the called unit has nested units, the status for the unit where the error occurred will be returned. There are no known error conditions within the units. A failed status code indicates that an unexpected Ada exception was raised. The only exception to this would be the Generic_List units which do raise exceptions and Generic_List_Utilities which has functions that may raise exceptions. (See Generic_List_Utilities for procedures which call these functions, trap the exceptions, and return a status code.)

A possible system error condition could be created by using the Smooth_Update_Position.Smooth_Entity unit and not deleting expired entities from the hash table this unit generates. The hash table is unbounded and could possibly use up more resources than are available or degrade performance if expired entities are not deleted from it. Hashing.Delete_Item unit should be called to delete expired entities.

The following is a list of all the units that return status values in alphabetial order according to the package and unit name. The status value and what could cause the error status is included. All units which execute with no error condition being trapped return a status value of DL_Status.SUCCESS.

Calculate.Az_And_El
DL_Status.CALC_AZ_AND_EL_FAILURE
Unknown Ada exception was raised.

Calculate.Azimuth
DL_Status.CALC_AZ_FAILURE
Unknown Ada exception was raised.

Calculate.Calculate_Azimuth
DL_Status.CALCULATE_AZ_FAILURE
Unknown Ada exception was raised.

Calculate.Calculate_Elevation
DL_Status.CALCULATE_EL_FAILURE
Unknown Ada exception was raised.

Calculate.Distance
DL_Status.CALC_DISTANCE_FAILURE
Unknown Ada exception was raised.

Calcuate.Elevation

DL_Status.CALC_EL_FAILURE
Unknown Ada exception was raised.

Calculate.Velocity

DL_Status.CALC_VELOCITY_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Entity_To_Geocentric_Conversion

DL_Status.ENT_GCC_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Entity_To_Geocentric_Vel_Conversion

DL_Status.ENT_GCC_VEL_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Geocentric_To_Entity_Conversion

DL_Status.GCC_ENT_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Geocentric_To_Entity_Vel_Conversion

DL_StatusGCC_ENT_VEL_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Geocentric_To_Geodetic_Conversion

DL_Status.GCC_GDC_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Geocentric_To_Local_Conversion

DL_Status.GCC_LOC_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Geocentric_To_Local_In_Meters_Conversion

DL_Status.GCC_LOC_M_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Geodetic_To_Geocentric_Conversion

DL_Status.GDC_GCC_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Geodetic_To_Local_Conversion

DL_Status.GDC_LOC_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Local_To_Geocentric_Conversion
DL_Status.LOC_GCC_FAILURE
Unknown Ada exception was raised.

Coordinate_Conversion.Local_To_Geodetic_Conversion
DL_Status.LOC_GDC_FAILURE
Unknown Ada exception was raised.

Dead_Reckoning.DRM_FPW
DL_Status.DR_FPW_FAILURE
Unknown Ada exception was raised.

Dead_Reckoning.DRM_FVW
DL_Status.DR_FVW_FAILURE
Unknown Ada exception was raised.

Dead_Reckoning.DRM_RPW
DL_Status.DR_RPW_FAILURE
Unknown Ada exception was raised.

Dead_Reckoning.DRM_RVW
DL_Status.DR_RVW_FAILURE
Unknown Ada exception was raised.

Dead_Reckoning.Rotate_Entity
DL_Status.ROTATE_FAILURE
Unknown Ada exception was raised.

Dead_Reckoning.Update_Position
DL_Status.DR_UDRP_FAILURE
Unknown Ada exception was raised.

DL_Math.Calculate_Euler_Trig
DL_Status.CAL_EULER_TRIG_FAILURE
Unknown Ada exception was raised.

DL_Math.Sin_Cos_Lat
DL_Status.SIN_COS_LAT_FAILURE
Unknown Ada exception was raised.

DL_Math.Sin_Con_Lon
DL_Status.SIN_COS_LON_FAILURE
Unknown Ada exception was raised.

Filter.Az_And_El
DL_Status.FILT_AZ_AND_EL_FAILURE
Unknown Ada exception was raised.

Filter.Azimuth
DL_Status.FILT_AZ_FAILURE
Unknown Ada exception was raised.

Filter.Distance
DL_Status.FILT_DIST_FAILURE
Unknown Ada exception was raised.

Filter.Elevation
DL_Status.FILT_EL_FAILURE
Unknown Ada exception was raised.

Filter.Maximum_Velocity
DL_Status.FILT_MAX_VEL_FAILURE
Unknown Ada exception was raised.

Filter.Minimum_Velocity
DL_Status.FILT_MIN_VEL_FAILURE
Unknown Ada exception was raised.

Filter_By_PDU_Components.Az_And_El
DL_Status.FILT_AZ_AND_EL_FAILURE
Unknown Ada exception was raised.

Filter_List.Entity_State_Az_And_El.Filter_Az_And_El *
DL_Status.FILT_AZ_ENT_ST_FAILURE
Unknown Ada exception was raised.

Filter_List.Detonation_Azimuth.Filter_Orientation
DL_Status.FILT_AZ_DETON_FAILURE
Unknown Ada exception was raised.

Filter_List.Entity_State_Azimuth.Filter_Orientation
DL_Status.FILT_AZ_ENT_ST_FAILURE
Unknown Ada exception was raised.

Filter_List.Fire_Azimuth.Filter_Orientation
DL_Status.FILT_AZ_FIRE_FAILURE
Unknown Ada exception was raised.

Filter_List.Detonation_Distance.Filter_Distance
DL_Status.FILT_DIST_DETON_FAILURE
Unknown Ada exception was raised.

Filter_List.Entity_State_Distance.Filter_Distance
DL_Status.FILT_DIST_ENT_ST_FAILURE
Unknown Ada exception was raised.

Filter_List.Fire_Distance.Filter_Distance
DL_Status.FILT_DIST_FIRE_FAILURE
Unknown Ada exception was raised.

Filter_List.Laser_Distance.Filter_Distance
DL_Status.FILT_DIST_LASER_FAILURE
Unknown Ada exception was raised.

Filter_List.Transmitter_Distance.Filter_Distance
DL_Status.FILT_DIST_TRANS_FAILURE
Unknown Ada exception was raised.

Filter_List.Detonation_Elevation.Filter_Orientation
DL_Status.FILT_EL_DETON_FAILURE
Unknown Ada exception was raised.

Filter_List.Entity_State_Elevation.Filter_Orientation
DL_Status.FILT_EL_ENT_ST_FAILURE
Unknown Ada exception was raised.

Filter_List.Fire_Elevation.Filter_Orientation
DL_Status.FILT_EL_FIRE_FAILURE
Unknown Ada exception was raised.

Filter_List.Entity_State_Max_Velocity.Filter_Velocity
DL_Status.FILT_MAX_VEL_ENT_ST_FAILURE
Unknown Ada exception was raised.

Filter_List.Entity_State_Min_Velocity.Filter_Velocity
DL_Status.FILT_MIN_VEL_ENT_ST_FAILURE
Unknown Ada exception was raised.

Hashing.Add_Item
DL_Status.ADD_ITEM_FAILURE
Unknown Ada exception was raised.

Hashing.Calculate_Hash_Table_Address

DL_Status.CAL_HASH_ADDRESS_FAILURE
Unknown Ada exception was raised.

Hashing.Delete_Item

DL_Status.DELETE_FREE_FAILURE
Unknown Ada exception was raised.

DL_Status.ITEM_NOT_FOUND
The Entity State PDU data was not in the hash table.

Hashing.Get_Item

DL_Status.GET_ITEM_FAILURE
Unknown Ada exception was raised.

DL_Status.ITEM_NOT_FOUND
The Entity State PDU data was not in the hash table.

Orientation_Conversions.Eulers_To_Local_Orientation

DL_Status.EUL_ORI_FAILURE
Unknown Ada exception was raised.

Orientation_Conversions.Locat_To_Eulers_Orientation

DL_Status.ORI_EUL_FAILURE
Unknown Ada exception was raised.

Smooth_Position_Update.Alpha_Beta_Filter

DL_Status.ALPHA_BETA_FILTER_FAILURE
Unknown Ada exception was raised.

Smooth_Position_Update.Rate_Change_Smoother

DL_Status.RATE_CHANGE_FAILURE
Unknown Ada exception was raised.

Smooth_Position_Update.Rate_Limiter

DL_Status.RATE_LIMITER_FAILURE
Unknown Ada exception was raised.

Smooth_Position_Update.Smooth_Entity

DL_Status.NO_FLAGS_SET
No smoothing algorithm is selected.

DL_Status.SMOOTH_ENTITY_FAILURE
Unknown Ada exception raised.

Sort_List.Detonation_Distance.Sort_Distance

DL_Status.LIST_IS_NULL
Indicates that the input list is null and cannot be sorted.

DL_Status.SORT_DIST_DETON_FAILURE
Unknown Ada exception was raised.

Sort_List.Entity_State_Distance.Sort_Distance
DL_Status.LIST_IS_NULL
Indicates that the input list is null and cannot be sorted.

DL_Status.SORT_DIST_ENT_ST_FAILURE
Unknown Ada exception was raised.

Sort_List.Fire_Distance.Sort_Distance
DL_Status.LIST_IS_NULL
Indicates that the input list is null and cannot be sorted.

DL_Status.SORT_DIST_FIRE_FAILURE
Unknown Ada exception was raised.

Sort_List.Laser_Distance.Sort_Distance
DL_Status.LIST_IS_NULL
Indicates that the input list is null and cannot be sorted.

DL_Status.SORT_DIST_LASER_FAILURE
Unknown Ada exception was raised.

Sort_List.Transmitter_Distance.Sort_Distance
DL_Status.LIST_IS_NULL
Indicates that the input list is null and cannot be sorted.

DL_Status.SORT_DIST_TRANS_FAILURE
Unknown Ada exception was raised.

Sort_List.Entity_State_Velocity.Sort_Velocity
DL_Status.LIST_IS_NULL
Indicates that the input list is null and cannot be sorted.

DL_Status.SORT_VEL_ENT_ST_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Append_List**
DL_Status.APPEND_LIST_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Assign_Item**
DL_Status.ASSIGN_ITEM_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Change_The_Item**
DL_Status.CHANGE_ITEM_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Check_At_End**
DL_Status.CHECK_AT_END_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Check_At_Head**
DL_Status.CHECK_AT_HEAD_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Check_List_Equal**
DL_Status.CHECK_LIST_EQUAL_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Check_Null**
DL_Status.CHECK_NULL_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Clear_Previous_Ptr**
DL_Status.CLEAR_PREVIOUS_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Clear_Next_Ptr**
DL_Status.CLEAR_NEXT_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Clear_The_List**
DL_Status.CLEAR_LIST_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Clear_The_Node**
DL_Status.CLEAR_NODE_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Copy_List**
DL_Status.COPY_LIST_FAILURE

Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Delete_Item**
DL_Status.DELETE_ITEM_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.
Delete_Item_And_Free_Storage
DL_Status.DELETE_FREE_ITEM_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Find_Position_Of_Item**
DL_Status.FIND_POSITION_OF_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Free_Node**
DL_Status.FREE_NODE_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Free_List**
DL_Status.FREE_LIST_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Get_First_Item**
DL_Status.GET_FIRST_ITEM_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Get_Last_Item**
DL_Status.GET_LAST_ITEM_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Get_Item**
DL_Status.GET_ITEM_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Get_Previous**
DL_Status.GET_PREVIOUS_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Get_Next**
DL_Status.GET_NEXT_FAILURE
Unknown Ada exception was raised.

{Instantiation of Generic_List_Utilities Package Name}.**Get_Sublist***
DL_Status.GET_SUBLIST_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Get_Size**
DL_Status.GET_SIZE_FAILURE

***{Instantiation of Generic_List_Utilities Package Name}.*Insert_Item**
DL_Status.INSERT_ITEM_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Insert_Item_End**
DL_Status.INSERT_END_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Insert_Item_Top**
DL_Status.INSERT_TOP_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Insert_List**
DL_Status.INSERT_LIST_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Split**
DL_Status.SPLIT_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Straight_Insertion_Sort**
DL_Status.SORT_INSERT_FAILURE
Unknown Ada exception was raised.

***{Instantiation of Generic_List_Utilities Package Name}.*Swap_Tails**
DL_Status.SWAP_TAILS_FAILURE
Unknown Ada exception was raised.

Generic_Sort_List_By_Distance.Create_Array_To_Sort
DL_Status.SORT_CREATE_ARRAY_FAILLURE
Unknown Ada exception was raised.

Generic_Sort_List_By_Distance.Create_Sorted_Linked_List
DL_Status.SORT_CREATE_LIST_FAILLURE
Unknown Ada exception was raised.

Generic_Sort_List_By_Distance.Sort_Distance
DL_Status.LIST_IS_NULL
Indicates that the input list is null and cannot be sorted.

DL_Status.*{Whatever status function is instantiated}*
Unknown Ada exception was raised.

Generic_Sort_List_By_Velocity.Create_Array_To_Sort
DL_Status.SORT_CREATE_ARRAY_FAILLURE
Unknown Ada exception was raised.

Generic_Sort_List_By_Velocity.Create_Sorted_Linked_List
DL_Status.SORT_CREATE_LIST_FAILLURE
Unknown Ada exception was raised.

Generic_Sort_List_By_Velocity.Sort_Velocity
DL_Status.LIST_IS_NULL
Indicates that the input list is null and cannot be sorted.

DL_Status.*{Whatever status function is instantiated}*
Unknown Ada exception was raised.

Vector_Math.Add_Offsets
DL_Status.ADD_OFFSETS_FAILURE
Unknown Ada exception was raised.

Vector_Math.Calculate_Vector_Difference
DL_Status.CALCULATE_DIFFERENCE_FAILURE
Unknown Ada exception was raised.

Vector_Math.Calculate_Vector_Offsets
DL_Status.CALCULATE_OFFSETS_FAILURE
Unknown Ada exception was raised.

Vector_Math.Check_Tolerance
DL_Status.CHECK_TOLERANCE_FAILURE
Unknown Ada exception was raised.

** Unit is overloaded (i.e., same name different parameters/implementation).*

The following units in Generic_List could raise exceptions. The exception and reason it would be raised is provided. These units should not be used without providing exception handlers within the calling unit to trap the raised exception.

***{Instantiation of Generic_List Package Name}.*Change_Item**
LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Clear_List**

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Clear_Previous**

LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Clear_Next**

LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Clear_Node**

LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Construct_Bottom**

OVERFLOW
The list cannot grow large enough to hold the item.

NOT_AT_END
The pointer does not point to the last item in the list.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Construct_Top**

OVERFLOW
The list cannot grow large enough to hold the item.

NOT_AT_HEAD
The pointer does not point to the first item in the list.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Copy**
OVERFLOW
The destination list cannot grow large enough to hold the source list.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Free**
GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Set_Head**
LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Swap_Tail**
NOT_AT_HEAD
The input pointer, The_List.Previous, is null.

LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Is_Equal**
GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Is_Null**
GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Length_Of**
GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Predecessor_Of**

LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Tail_Of**

LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

***{Instantiation of Generic_List Package Name}.*Value_Of**

LIST_IS_NULL
The input pointer is null.

GEN_LIST_FAILURE
An unknown Ada exception was raised.

Appendixtc "**Appendix**"\l 1§

Package/Unit Overview tc " Package/Unit Overview"\l 2§
(For callable units)

Calculate Package - provides units to calculate azimuth, elevation, distance and velocity.

Elevation or azimuth is calculated from the input vector position.

- Calculate.Calculate_Azimuth
- Calculate.Calculate_Elevation

Elevation/azimuth calculated in reference to an input Entity State PDU position. The position of interest is converted to the Entity State PDU position's coordinate system.

- Calculate.Az_And_El
- Calculate.Azimuth
- Calcuate.Elevation

Distance is calculated from two vector position inputs.

- Calculate.Distance

Velocity magnitude is calculated from a velocity vector input.

- Calculate.Velocity

Coordinate_Conversion Package - provides units to convert between various coordinate systems such as geocentric, geodetic, entity and local.

Converts using Euler Angles, world coordinates, and entity or velocity coordinates.

- Coordinate_Conversion.Entity_To_Geocentric_Conversion
- Coordinate_Conversion.Entity_To_Geocentric_Vel_Conversion
- Coordinate_Conversion.Geocentric_To_Entity_Conversion
- Coordinate_Conversion.Geocentric_To_Entity_Vel_Conversion

Converts using world coordinates and latitude, longitude, and altitude.

- Coordinate_Conversion.Geocentric_To_Geodetic_Conversion
- Coordinate_Conversion.Geodetic_To_Geocentric_Conversion

Converts using local coordinates in feet/meters and world coordinates in meters.

- Coordinate_Conversion.Geocentric_To_Local_Conversion
- Coordinate_Conversion.Geocentric_To_Local_In_Meters_Conversion
- Coordinate_Conversion.Local_To_Geocentric_Conversion

Converts using local coordinates in feet and latitude, longitude, and altitude.

- Coordinate_Conversion.Geodetic_To_Local_Conversion
- Coordinate_Conversion.Local_To_Geodetic_Conversion

Dead-Reckoning Package - provides a unit to dead-reckon an entity according to the following DIS dead-reckoning models: FPW, FVW, RPW, RVW, and Static (i.e., any model not listed is dead-reckoned with the Static model).

Input is elapsed time in microseconds and an Entity_State_PDU, outputs Entity_State_PDU with position data updated.

- Dead_Reckoning.Update_Position

Filter/Filter_By_PDU_Component Packages - provides units which detemine if a PDU's data is within a predefined threshold.

Azimuth/elevation threshold ranges, reference Entity State PDU or location and orientation of entity; and position of interest is input. A boolean value as to whether the position of interest is within the threshold range is output.

- Filter.Az_And_El
- Filter_By_PDU_Components.Az_And_El
- Filter.Azimuth
- Filter.Elevation

Two positions and a maximum distance threshold is input and a boolean value as to whether the distance between the two positions exceeds the threshold is output.

- Filter.Distance

A velocity vectory and maximum/minimum velocity threshold is input and a boolean value as to whether the velocity magnitude of the input velocity vector exceeds the threshold is output.

- Filter.Maximum_Velocity
- Filter.Minimum_Velocity

Filter_List Package - instantiates generic filter packages which filter a list of PDUs based on some specified criteria and threshold. Units that do not meet the filter criteria are deleted from the list.

Inputs are a list of pointers to PDUs and threshold ranges. The input list is output with the PDUs which do not meet the specified criteria deleted.

- Filter_List.Entity_State_Az_And_El.Filter_Az_And_El *
- Filter_List.Detonation_Azimuth.Filter_Orientation
- Filter_List.Entity_State_Azimuth.Filter_Orientation
- Filter_List.Fire_Azimuth.Filter_Orientation
- Filter_List.Detonation_Distance.Filter_Distance
- Filter_List.Entity_State_Distance.Filter_Distance
- Filter_List.Fire_Distance.Filter_Distance
- Filter_List.Laser_Distance.Filter_Distance
- Filter_List.Transmitter_Distance.Filter_Distance
- Filter_List.Detonation_Elevation.Filter_Orientation
- Filter_List.Entity_State_Elevation.Filter_Orientation
- Filter_List.Fire_Elevation.Filter_Orientation
- Filter_List.Entity_State_Max_Velocity.Filter_Velocity
- Filter_List.Entity_State_Min_Velocity.Filter_Velocity

** Unit is overloaded (i.e., same name different parameters/implementation)*

Hashing Package - contains units to maintain the hash table used by the Smooth_Position_Update package.

Input is an Entity_State_PDU.

- Hashing.Delete_Item

Orientation_Conversions Package - contains units to convert between Euler Angles and local roll, pitch, and heading; and vice versa.

Euler_Angles, latitude, longitude, altitude and local orientation (roll, pitch, and heading) are used to do the conversions.

- Orientation_Conversions.Eulers_To_Local_Orientation
- Orientation_Conversions.Locat_To_Eulers_Orientation

Smooth_Position_Update Package - contains a unit which dead-reckons and smooths entities. One of the following three smoothers may be called by executing this unit: Alpha_Beta_Filter, Rate_Limiter, Rate_Change_Smoother.

An Entity_State_PDU for the entity to be smoothed/dead-reckoned, elapsed time since last update, the tolerance or coefficient value and the smoother algorithm to implement are input. The Entity_State_PDU with the updated position data is output.

- Smooth_Position_Update.Smooth_Entity

Sort_List Package - instantiates generic binary insertion sort packages which sorts a list of PDUs based on some specified criteria.

A list of pointers to PDUs in input along with the sort criteria and sorting order (i.e, ascending, descending). The list is output in sorted order.

- Sort_List.Detonation_Distance.Sort_Distance
- Sort_List.Entity_State_Distance.Sort_Distance
- Sort_List.Fire_Distance.Sort_Distance
- Sort_List.Laser_Distance.Sort_Distance
- Sort_List.Transmitter_Distance.Sort_Distance
- Sort_List.Entity_State_Velocity.Sort_Velocity

Generic_Binary_Insertion_Sort Package - provides a generic package which can be instantiated for any data type and any sort criteria.

See Generic_Sort_List_By_Distance and Generic_Sort_List_By_Velocity Packages for current instantiations.

- *{Instantiation Package Name}.*Sort

Generic_Filter_List_By_Az_And_El Package - provides a generic package to evaluate each PDU in a list and remove those PDUs which do not meet the specified azimuth and elevation thresholds. This unit can be instantiated for any PDU type

See Filter_List Package for current instantiations

- *{Instantiation Package Name}.*Filter_Az_And_El*

** Unit is overloaded (i.e., same name different parameters/implementation)*

Generic_Filter_List_By_Distance Package -provides a generic package to evaluate each PDU in a list and remove those PDUs which do not meet the specified distance threshold. This unit can be instantiated for any PDU type.

See Filter_List Package for current instantiations

- *{Instantiation Package Name}.*Filter_Distance

Generic_Filter_List_By_Orientation Package - provides a generic package to evaluate each PDU in a list and remove those PDUs which do not meet the specified azimuth or elevation threshold. This unit can be instantiated for any PDU type and either azimuth or elevation orientation.

See Filter_List Package for current instantiations

- *{Instantiation Package Name}.*Filter_Orientation

Generic_Filter_List_By_Velocity Package - provides a generic package to evaluate each PDU in a list and remove those PDUs which do not meet the specified minimum or maximum velocity threshold. This unit can be instantiated for any PDU type and either minimum or maximum velocity.

See Filter_List Package for current instantiations

- *{Instantiation Package Name}.*Filter_Velocity

Generic_List Package - provides units to create and manage a generic double linked, unbounded list of zero or more items in which items can be added to and removed from any position such that a strict linear ordering is maintained.

This package may be instantiated for any data type. (These units will raise exceptions if errors are encountered. See Generic_List_Utilities for the same units which return status codes instead of raising exceptions and other units which use the same data structure.)

See DL_Linked_List Package for current instantiations

- *{Instantiation Package Name}.*Change_Item
- *{Instantiation Package Name}.*Clear_List
- *{Instantiation Package Name}.*Clear_Previous
- *{Instantiation Package Name}.*Clear_Next
- *{Instantiation Package Name}.*Clear_Node
- *{Instantiation Package Name}.*Construct_Bottom
- *{Instantiation Package Name}.*Construct_Top
- *{Instantiation Package Name}.*Copy
- *{Instantiation Package Name}.*Free
- *{Instantiation Package Name}.*Set_Head
- *{Instantiation Package Name}.*Swap_Tail
- *{Instantiation Package Name}.*Is_Equal
- *{Instantiation Package Name}.*Is_Null
- *{Instantiation Package Name}.*Length_Of
- *{Instantiation Package Name}.*Predecessor_Of
- *{Instantiation Package Name}.*Tail_Of
- *{Instantiation Package Name}.*Value_Of

Generic_List_Utilities Package - is built on the more primative Generic_List units which it imports. When this package is instantiated the following units are

available.

- *{Instantiation Package Name}.*Append_List
- *{Instantiation Package Name}.*Assign_Item
- *{Instantiation Package Name}.*Change_The_Item
- *{Instantiation Package Name}.*Check_At_End
- *{Instantiation Package Name}.*Check_At_Head
- *{Instantiation Package Name}.*Check_List_Equal
- *{Instantiation Package Name}.*Check_Null
- *{Instantiation Package Name}.*Clear_Previous_Ptr
- *{Instantiation Package Name}.*Clear_Next_Ptr
- *{Instantiation Package Name}.*Clear_The_List
- *{Instantiation Package Name}.*Clear_The_Node
- *{Instantiation Package Name}.*Copy_List
- *{Instantiation Package Name}.*Delete_Item
- *{Instantiation Package Name}.*Delete_Item_And_Free_Storage
- *{Instantiation Package Name}.*Find_Position_Of_Item
- *{Instantiation Package Name}.*Free_Node
- *{Instantiation Package Name}.*Free_List
- *{Instantiation Package Name}.*Get_First_Item
- *{Instantiation Package Name}.*Get_Last_Item
- *{Instantiation Package Name}.*Get_Item
- *{Instantiation Package Name}.*Get_Previous
- *{Instantiation Package Name}.*Get_Next
- *{Instantiation Package Name}.*Get_Sublist*
- *{Instantiation Package Name}.*Get_Size
- *{Instantiation Package Name}.*Insert_Item
- *{Instantiation Package Name}.*Insert_Item_End
- *{Instantiation Package Name}.*Insert_Item_Top
- *{Instantiation Package Name}.*Insert_List
- *{Instantiation Package Name}.*Split
- *{Instantiation Package Name}.*Straight_Insertion_Sort
- *{Instantiation Package Name}.*Swap_Tails
- *{Instantiation Package Name}.*End_Of
- *{Instantiation Package Name}.*Head_Of
- *{Instantiation Package Name}.*Is_End
- *{Instantiation Package Name}.*Is_Head
- *{Instantiation Package Name}.*Location_Of*
- *{Instantiation Package Name}.*Position_Of

** Unit is overloaded (i.e., same name different parameters/implementation)*
Generic_Sort_List_By_Distance Package - is a generic package which instantiates the Generic_Binary_Insertion_Sort for distance. This package can be instantiated for any PDU type.

See Sort_List for current instantiations.

- *{Instantiation Package Name}.*Sort_Distance

Generic_Sort_List_By_Velocity Package - is a generic package which instantiates the Generic_Binary_Insertion_Sort for velocity. This package can be instantiated for any PDU type.

See Sort_List for current instantiations.

- *{Instantiation Package Name}.*Sort_Velocity

Package and Unit Name to Filename
Cross Reference Tabletc "Package and Unit Name to Filename Cross Reference Table" \f t§

Package/Unit Name	Filename
Calculate *{Package Specification}*	Calculate_.ada
Calculate *{Package Body}*	Calculate.ada
{Calculate Package units which are separate}	
Calculate.Az_And_El	Cal__Az_And_El.ada
Calculate.Azimuth	Cal__Azimuth.ada
Calculate.Calculate_Azimuth	Cal__Calculate_Azimuth.ada
Calculate.Calculate_Elevation	Cal__Calculate_Elevation.ada
Calculate.Distance	Cal__Distance.ada
Calculate.Elevation	Cal__Elevaton.ada
Calculate.Velocity	Cal__Velocity.ada
Coordinate_Conversion *{Package Specification}*	Coordinate_Conversion_.ada
Coordinate_Conversion *{Package Body}*	Coordinate_Conversion.ada

{Coordinate_Conversion Package units which are separate}	
Coordinate_Conversion. Entity_To_Geocentric_Conversion	CC__Entity_To_Geocentric_Conversion.ada
Coordinate_Conversion. Entity_To_Geocentric_Vel_Conversion	CC__Entity_To_Geocentric_Vel_Conversion.ada
Coordinate_Conversion. Geocentric_To_Entity_Conversion	CC__Geocentric_To_Entity_Conversion.ada
Coordinate_Conversion. Geocentric_To_Entity_Vel_Conversion	CC__Geocentric_To_Entity_Vel_Conversion.ada
Coordinate_Conversion. Geocentric_To_Geodetic_Conversion	CC__Geocentric_To_Geodetic_Conversion.ada
Coordinate_Conversion. Geocentric_To_Local_Conversion	CC__Geocentric_To_Local_Conversion.ada
Coordinate_Conversion. Geocentric_To_Local_In_Meters _Conversion	CC__Geocentric_To_Local_In_Meters_ Conversion.ada
Coordinate_Conversion. Geodetic_To_Geocentric_Conversion	CC__Geodetic_To_Geocentric_Conversion.ada
Coordinate_Conversion. Geodetic_To_Local_Conversion	CC__Geodetic_To_Local_Conversion.ada
Coordinate_Conversion. Local_To_Geocentric_Conversion	CC__Local_To_Geocentric_Conversion.ada

Coordinate_Conversion. Local_To_Geodetic_Conversion	CC__Local_To_Geodetic_Conversion.ada
Dead_Reckoning *{Package Specification}*	Dead_Reckoning_.ada
Dead_Reckoning *{Package Body}*	Dead_Reckoning.ada
{Dead_Reckoning units}	Dead_Reckoning.ada
Dead_Reckoning.DRM_FPW#	Dead_Reckoning.ada
Dead_Reckoning.DRM_FVW#	Dead_Reckoning.ada
Dead_Reckoning.DRM_RPW#	Dead_Reckoning.ada
Dead_Reckoning.DRM_RVW#	Dead_Reckoning.ada
Dead_Reckoning.Rotate_Entity#	Dead_Reckoning.ada
Dead_Reckoning.Update_Position	Dead_Reckoning.ada
DL_Linked_List_Types *{Package Specification}*	DL_Linked_List_Types_.ada
DL_Linked_List_Types *{Package Body}*	DL_Linked_List_Types.ada

{DL_Linked_List_Types instantiations of Generic_List and Generic_List_Utilities units (i.e. all units in these packages are callable with the following instantiation prefixes.)}	
DL_Linked_List_Types. Detonation_List *(Instantiates Generic_List units)*	DL_Linked_List_Types_.ada
DL_Linked_List_Types. Detonation_List_Utilities *(Instantiates Generic_List_Utilities units)*	DL_Linked_List_Types_.ada
DL_Linked_List_Types. Entity_State_List *(Instantiates Generic_List units)*	DL_Linked_List_Types_.ada
DL_Linked_List_Types. Entity_State_List_Utilities *(Instantiates Generic_List_Utilities units)*	DL_Linked_List_Types_.ada
DL_Linked_List_Types. Fire_List *(Instantiates Generic_List units)*	DL_Linked_List_Types_.ada
DL_Linked_List_Types. Fire_List_Utilities *(Instantiates Generic_List_Utilities units)*	DL_Linked_List_Types_.ada
DL_Linked_List_Types. Laser_List *(Instantiates Generic_List units)*	DL_Linked_List_Types_.ada
DL_Linked_List_Types. Laser_List_Utilities *(Instantiates Generic_List_Utilities units)*	DL_Linked_List_Types_.ada
DL_Linked_List_Types.	DL_Linked_List_Types_.ada

Transmitter_List *(Instantiates Generic_List units)*	
DL_Linked_List_Types. Transmitter_List_Utilities *(Instantiates Generic_List_Utilities units)*	DL_Linked_List_Types_.ada
DL_Math *{Package Specification}*	DL_Math_.ada
DL_Math*{Package Body}*	DL_Math.ada
{DL_Math Package units}	DL_Math.ada
DL_Math.Calculate_Euler_Trig*#	DL_Math.ada
DL_Math.Sin_Cos_Lat*#	DL_Math.ada
DL_Math.Sin_Cos_Lon*#	DL_Math.ada
Filter *{Package Specification}*	Filter_.ada
Filter *{Package Body}*	Filter.ada
{Filter Package units which are separate}	
Filter.Azimuth	Filter__Azimuth.ada

Filter.Distance	Filter__Distance.ada
Filter.Elevation	Filter__Elevation.ada
Filter.Maximum_Velocity	Filter__Maximum_Velocity.ada
Filter.Minimum_Velocity	Filter__Minimum_Velocity.ada
Filter_By_PDU_Components *{Package Specification}*	Filter_By_PDU_Components_.ada
Filter_By_PDU_Components *{Package Body}*	Filter_By_PDU_Components_.ada
{Filter_By_PDU_Components Package units which are separate}	Filter_By_PDU_Components.ada
Filter_By_PDU_Components.Az_And_El	Filter_By_PDU_Cmpnts__Az_And_El.ada
Filter_List *{Package Specification}*	Filter_List_.ada
{Filter_List Package callable units (these are instantiations of generic filter units)}	
Filter_List.Entity_State_Az_And_El. Filter_Az_And_El *	Filter_List_.ada
Filter_List. Detonation_Azimuth.Filter_Orientation	Filter_List_.ada

Filter_List. Entity_State_Azimuth.Filter_Orientation	Filter_List_.ada
Filter_List. Fire_Azimuth.Filter_Orientation	Filter_List_.ada
Filter_List. Detonation_Distance.Filter_Distance	Filter_List_.ada
Filter_List. Entity_State_Distance.Filter_Distance	Filter_List_.ada
Filter_List. Fire_Distance.Filter_Distance	Filter_List_.ada
Filter_List. Laser_Distance.Filter_Distance	Filter_List_.ada
Filter_List. Transmitter_Distance.Filter_Distance	Filter_List_.ada
Filter_List. Detonation_Elevation.Filter_Orientation	Filter_List_.ada
Filter_List. Entity_State_Elevation.Filter_Orientation	Filter_List_.ada
Filter_List. Fire_Elevation.Filter_Orientation	Filter_List_.ada
Filter_List. Entity_State_Max_Velocity. Filter_Velocity	Filter_List_.ada

Filter_List. Entity_State_Min_Velocity.Filter_Velocity	Filter_List_.ada
Generic_Binary_Insertion_Sort *{Generic Package Specification}*	Generic_Binary_Insertion_Sort.ada
Generic_Binary_Insertion_Sort *{ Generic Package Body}*	Generic_Binary_Insertion_Sort_.ada
{Generic Binary Insertion Sort Package unit}	
*{Instantiation Package Name}.*Sort	Generic_Binary_Insertion_Sort.ada
Generic_Filter_List_By_Az_And_El *{Package Specification}*	Generic_Filter_List_By_Az_And_El_.ada
Generic_Filter_List_By_Az_And_El*{Package Body}*	Generic_Filter_List_By_Az_And_El.ada
{Generic_Filter_List_By_Az_And_El Package unit}	
{Instantiation Package Name}. Filter_Az_And_El*	Generic_Filter_List_By_Az_And_El.ada

Generic_Filter_List_By_Distance *{Package Specification}*	Generic_Filter_List_By_Distance_.ada
Generic_Filter_List_By_Distance*{Package Body}*	Generic_Filter_List_By_Distance.ada
{Generic_Filter_List_By_Distancel Package unit}	
{Instantiation Package Name}. Filter_Distance	Generic_Filter_List_By_Distance.ada
Generic_Filter_List_By_Orientation *{Package Specification}*	Generic_Filter_List_By_Orientation_.ada
Generic_Filter_List_By_Orientation*{Package Body}*	Generic_Filter_List_By_Orientation.ada
{Generic_Filter_List_By_Orientation Package unit}	
{Instantiation Package Name}. Filter_Orientation	Generic_Filter_List_By_Orientation.ada
Generic_Filter_List_By_Velocity *{Package Specification}*	Generic_Filter_List_By_Velocity_.ada
Generic_Filter_List_By_Velocity *{Package Body}*	Generic_Filter_List_By_Velocity.ada

{Generic_Filter_List_By_Velocity package unit }	
{Instantiation Package Name}. Filter_Velocity	Generic_Filter_List_By_Velocity.ada
Generic_List *{Package Specification}*	Generic_List_.ada
Generic_List *{Package Body}*	Generic_List.ada
{Generic_List Package units}	
*{Instantiation Package Name}.*Change_Item	Generic_List.ada
*{Instantiation Package Name}.*Clear_List	Generic_List.ada
{Instantiation Package Name}. Clear_Previous	Generic_List.ada
*{Instantiation Package Name}.*Clear_Next	Generic_List.ada
*{Instantiation Package Name}.*Clear_Node	Generic_List.ada
{Instantiation Package Name}. Construct_Bottom	Generic_List.ada
{Instantiation Package Name}. Construct_Top	Generic_List.ada

*{Instantiation Package Name}.*Copy	Generic_List.ada
*{Instantiation Package Name}.*Free	Generic_List.ada
*{Instantiation Package Name}.*Set_Head	Generic_List.ada
*{Instantiation Package Name}.*Swap_Tail	Generic_List.ada
*{Instantiation Package Name}.*Is_Equal	Generic_List.ada
*{Instantiation Package Name}.*Is_Null	Generic_List.ada
*{Instantiation Package Name}.*Length_Of	Generic_List.ada
{Instantiation Package Name}. Predecessor_Of	Generic_List.ada
*{Instantiation Package Name}.*Tail_Of	Generic_List.ada
*{Instantiation Package Name}.*Value_Of	Generic_List.ada
Generic_List_Utilities *{Package Specification}*	Generic_List_Utilities_.ada
Generic_List_Utilities *{Package Body}*	Generic_List_Utilities.ada
{Generic_List_UtilitiesPackage units}	
*{Instantiation Package Name}.*Append_List	Generic_List_Utilities.ada

*{Instantiation Package Name}.*Assign_Item	Generic_List_Utilities.ada
{Instantiation Package Name}. Change_The_Item	Generic_List_Utilities.ada
{Instantiation Package Name}. Check_At_End	Generic_List_Utilities.ada
{Instantiation Package Name}. Check_At_Head	Generic_List_Utilities.ada
{Instantiation Package Name}. Check_List_Equal	Generic_List_Utilities.ada
{Instantiation Package Name}. Check_Null	Generic_List_Utilities.ada
{Instantiation Package Name}. Clear_Previous_Ptr	Generic_List_Utilities.ada
{Instantiation Package Name}. Clear_Next_Ptr	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Clear_The_List	Generic_List_Utilities.ada
{Instantiation Package Name}. Clear_The_Node	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Copy_List	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Delete_Item	Generic_List_Utilities.ada

{Instantiation Package Name}. Delete_Item_And_Free_Storage	Generic_List_Utilities.ada
{Instantiation Package Name}. Find_Position_Of_Item	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Free_Node	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Free_List	Generic_List_Utilities.ada
{Instantiation Package Name}. Get_First_Item	Generic_List_Utilities.ada
{Instantiation Package Name}. Get_Last_Item	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Get_Item	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Get_Previous	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Get_Next	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Get_Sublist*	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Get_Size	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Insert_Item	Generic_List_Utilities.ada
{Instantiation Package Name}. Insert_Item_End	Generic_List_Utilities.ada

{Instantiation Package Name}. Insert_Item_Top	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Insert_List	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Split	Generic_List_Utilities.ada
{Instantiation Package Name}. Straight_Insertion_Sort	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Swap_Tails	Generic_List_Utilities.ada
*{Instantiation Package Name}.*End_Of	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Head_Of	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Is_End	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Is_Head	Generic_List_Utilities.ada
{Instantiation Package Name}. Location_Of*	Generic_List_Utilities.ada
*{Instantiation Package Name}.*Position_Of	Generic_List_Utilities.ada
Generic_Sort_List_By_Distance *{Generic Package Specification}*	Generic_Sort_List_By_Distance_.ada
Generic_Sort_List_By_Distance *{ Generic Package Body}*	Generic_Sort_List_By_Distance.ada

{Generic_Sort_List_By_Distance Package unit}	
{Instantiation Package Name}. Sort_Distance	Generic_Sort_List_By_Distance.ada
Generic_Sort_List_By_Velocity *{Package Specification}*	Generic_Sort_List_By_Velocity_.ada
Generic_Sort_List_By_Velocity *{Package Body}*	Generic_Sort_List_By_Velocity.ada
{Generic_Sort_List_By_Velocity Package unit}	
{Instantiation Package Name}. Sort_Velocity	Generic_Sort_List_By_Velocity.ada
Hashing *{Package Specification}*	Hashing_.ada
Hashing *{Package Body}*	Hashing.ada
{Hashing Package units}	
Hashing.Add_Item#	Hashing.ada
Hashing.Calculate_Hash_Table_Address#	Hashing.ada

Hashing.Delete_Item	Hashing.ada
Hashing.Get_Item#	Hashing.ada
Orientation_Conversions *{Package Specification}*	Orientation_Conversions_.ada
Orientation_Conversions *{Package Body}*	Orientation_Conversions.ada
{Orientation_Conversions units}	
Orientation_Conversions. Eulers_To_Local_Orientation	Orientation_Conversion.ada
Orientation_Conversions. Local_To_Eulers_Orientation	Orientation_Conversion.ada
Smooth_Position_Update *{Package Specification}*	Smooth_Position_Update_.ada
Smooth_Position Update *{Package Body}*	Smooth_Position_Update.ada
{Smooth_Position units}	
Smooth_Position_Update. Alpha_Beta_Filter#	SPU__Smooth_Entity.ada

Smooth_Position_Update. Rate_Change_Smoother#	SPU__Smooth_Entity.ada
Smooth_Position_Update.Rate_Limiter#	SPU__Smooth_Entity.ada
Smooth_Position_Update.Smooth_Entity#	SPU__Smooth_Entity.ada
Sort_List *{Package Specification}*	Sort_List_.ada
{Sort_List instantiations of Generic_Binary_Insertion_Sort units}	
Sort_List.Detonation_Distance. Sort_Distance	Sort_List_.ada
Sort_List.Entity_State_Distance. Sort_Distance	Sort_List_.ada
Sort_List.Fire_Distance.Sort_Distance	Sort_List_.ada
Sort_List.Laser_Distance.Sort_Distance	Sort_List_.ada
Sort_List.Transmitter_Distance. Sort_Distance	Sort_List_.ada
Sort_List.Entity_State_Velocity. Sort_Velocity	Sort_List_.ada

Vector_Math *{Package Specification}*	Vector_Math_.ada
Vector_Math *{Package Body}*	Vector_Math.ada
{Vector_Math Package units}	Vector_Math.ada
Vector_Math.Add_Offsets*#	Vector_Math.ada
Vector_Math. Calculate_Vector_Difference*#	Vector_Math.ada
Vector_Math.Calclate_Vector_Offsets*#	Vector_Math.ada
Vector_Math.Check_Tolerance*#	Vector_Math.ada

** Unit is overloaded (i.e., same name different parameters/implementation).*
 # Unit returns a status value that may need to be tracked if an error occurrs, but it is primarily a support routine for other units.