11 Application program interface

11.1 Introduction

This clause specifies an API for the operations and concepts defined in this International Standard. It specifies:

a) data types,

b) classes and their methods, and

c) functions.

The data types specified in this clause are composed of basic and structured data types. Data types supporting methods and functions are defined in 11.2. To support the conformance of exchange formats (see 14.3), additional data types for storage and/or transmission are defined in 11.5.

Class specifications serve to organize methods related to specific SRM concepts. In this sense, *class instances* represent SRM concept instances. An API object is an instance of a class. A *class* defines methods that produce outputs by operating on the *state* of an object and its inputs. Classes and their methods are defined in 11.3.

Functions are specified outside of the class specifications and operate only on the specified inputs to produce their corresponding outputs. The capabilities provided by these functions include creating instances of standard and set-based SRFs, and querying the extent of support of an API implementation. These functions are specified in 11.4.

11.2 Data types

11.2.1 Overview

Data types are organized into *basic data types* and *structured data types*. Basic data types consist of single values, whereas structured data types consist of multiple values. Basic data types include numeric data types, enumerated data types, and selection data types. Selection data types are similar to enumerated data types, but can be extended via registration. Structured data types include array data types, record data types and variant record data types. The elements of arrays are all of the same data type and are referenced by position within the array, whereas the elements of records may be of different data types and are referenced by name. In variant records, a selector is used to choose one record data type from among several alternative record data types.

11.2.2 Abbreviations

Table 11.1 lists the SRFTs and their abbreviations used in the formation of enumerant names and record element names of data types.

Table 11.1 — SRFT abbreviations

Abbreviation	SRFT
CC	Celestiocentric
CD	Celestiodetic
CM	Celestiomagnetic

Abbreviation	SRFT
EC	Equidistant Cylindrical
EI	Equatorial Inertial
HAEC	Heliospheric Aries Ecliptic
HEEC	Heliospheric Earth Ecliptic
HEEQ	Heliospheric Earth Equatorial
LCC	Lambert Conformal Conic
LCE_3D	Lococentric Euclidean 3D
LSA	Local Space Azimuthal
LSP	Local Space Polar
LSR_2D	Local Space Rectangular 2D
LSR_3D	Local Space Rectangular 3D
LTSAS	Local Tangent Space Azimuthal Spherical
LTSC	Local Tangent Space Cylindrical
LTSE	Local Tangent Space Euclidean
M	Mercator
OMS	Oblique Mercator Spherical
PD	Planetodetic
PS	Polar Stereographic
SEC	Solar Ecliptic
SEQ	Solar Equatorial
SMD	Solar Magnetic Dipole
SME	Solar Magnetic Ecliptic
TM	Transverse Mercator

11.2.3 Numbers

Two categories of numbers are specified: integer numbers and floating-point numbers. The general-purpose integer data types are `Integer`, `Integer_Positive` and `Integer_Unsigned`. All implementations that conform to this standard shall support at least the minimum ranges for values of these data types as specified in Table 11.2.

Table 11.2 — Integer data types

Data type	Value range
`Integer`	[-2 147 483 647, 2 147 483 647]
`Integer_Positive`	[1, 4 294 967 295]
`Integer_Unsigned`	[0, 4 294 967 295]

`Long_Float` is a data type defined for floating-point numbers. This data type corresponds to the double precision floating-point data type specified by ISO/IEC/IEEE 60559:2011. However, implementations on architectures that support other floating-point representations are allowed. When recording a `Long_Float` number in a file or archive, the floating-point data type specified in ISO/IEC/IEEE 60559:2011 shall be used. It is the responsibility of the implementation to make suitable conversions when the internal floating point format differs from the standard floating point data type.

11.2.4 Logicals

The general-purpose logical data type is Boolean. All implementations that conform to this standard shall support this data type as specified in Table 11.3.

Table 11.3 — Logical data type

Data type	Values
`Boolean`	[false (or 0), true (or 1)]

11.2.5 Object_Reference

An *object reference* is a generic reference to a class instance. `Object_Reference` is an opaque data type that implements this concept. If the values of two `Object_Reference`s are equal, they shall refer to the same class instance. In all the method specifications in this clause, whenever an argument passed to or returned from a method is a class instance, it is an `Object_Reference` that is passed or returned.

The `NULL_Object` is defined as a special `Object_Reference`. If the value of an `Object_Reference` is equal to the value of the `NULL_Object`, it does not reference any class instance. On an error condition, some language bindings may require method and/or function outputs to be defined. In these cases, `Object_Reference` outputs shall be set to `NULL_Object` as appropriate.

11.2.6 Enumerated data types

11.2.6.1 Introduction

Enumerated data types are data types whose values are specified from an ordered list of names. Enumerated data types are a closed list, the members of which do not change based on registration or deprecation.

11.2.6.2 Axis_Direction_2D

This data type represents the values of the axis direction parameter(s) of the SRFT LOCAL_SPACE_RECTANGULAR_2D.

```
Axis_Direction_3D ::= (   POSITIVE_PRIMARY_AXIS_2D,
                          POSITIVE_SECONDARY_AXIS_2D,
                          NEGATIVE_PRIMARY_AXIS_2D,
                          NEGATIVE_SECONDARY_AXIS_2D )
```

POSITIVE_PRIMARY_AXIS_2D indicates that the forward axis of the LOCAL_SPACE_RECTANGULAR_2D SRF is to be aligned with the positive primary axis of its LOCOCENTRIC_EUCLIDEAN_2D CS.

POSITIVE_SECONDARY_AXIS_2D indicates that the forward axis of the LOCAL_SPACE_RECTANGULAR_2D SRF is to be aligned with the positive secondary axis of its LOCOCENTRIC_EUCLIDEAN_2D CS.

NEGATIVE_PRIMARY_AXIS_2D indicates that the forward axis of the LOCAL_SPACE_RECTANGULAR_2D SRF is to be aligned with the negative primary axis of its LOCOCENTRIC_EUCLIDEAN_2D CS.

NEGATIVE_SECONDARY_AXIS_2D indicates that the forward axis of the LOCAL_SPACE_RECTANGULAR_2D SRF is to be aligned with the negative secondary axis of its LOCOCENTRIC_EUCLIDEAN_2D CS.

11.2.6.3 Axis_Direction_3D

This data type represents the values of the axis direction parameter(s) of the SRFT LOCAL_SPACE_RECTANGULAR_3D.

```
Axis_Direction_3D ::= (   POSITIVE_PRIMARY_AXIS_3D,
                          POSITIVE_SECONDARY_AXIS_3D,
                          POSITIVE_TERTIARY_AXIS_3D,
                          NEGATIVE_PRIMARY_AXIS_3D,
                          NEGATIVE_SECONDARY_AXIS_3D,
                          NEGATIVE_TERTIARY_AXIS_3D )
```

POSITIVE_PRIMARY_AXIS_3D indicates that the specified (forward or up) axis of the LOCAL_SPACE_RECTANGULAR_3D SRF is to be aligned with the positive primary axis of its LOCOCENTRIC_EUCLIDEAN_3D CS.

POSITIVE_SECONDARY_AXIS_3D indicates that the specified (forward or up) axis of the LOCAL_SPACE_RECTANGULAR_3D SRF is to be aligned with the positive secondary axis of its LOCOCENTRIC_EUCLIDEAN_3D CS.

POSITIVE_TERTIARY_AXIS_3D indicates that the specified (forward or up) axis of the LOCAL_SPACE_RECTANGULAR_3D SRF is to be aligned with the positive tertiary axis of its LOCOCENTRIC_EUCLIDEAN_3D CS.

NEGATIVE_PRIMARY_AXIS_3D indicates that the specified (forward or up) axis of the LOCAL_SPACE_RECTANGULAR_3D SRF is to be aligned with the negative primary axis of its LOCOCENTRIC_EUCLIDEAN_3D CS.

NEGATIVE_SECONDARY_AXIS_3D indicates that the specified (forward or up) axis of the LOCAL_SPACE_RECTANGULAR_3D SRF is to be aligned with the negative secondary axis of its LOCOCENTRIC_EUCLIDEAN_3D CS.

NEGATIVE_TERTIARY_AXIS_3D indicates that the specified (forward or up) axis of the LOCAL_SPACE_RECTANGULAR_3D SRF is to be aligned with the negative tertiary axis of its LOCOCENTRIC_EUCLIDEAN_3D CS.

11.2.6.4 Interval_Type

This data type is used to specify coordinate-component intervals.

```
Interval_Type::= (   OPEN_INTERVAL,
                     GE_LT_INTERVAL,
                     GT_LE_INTERVAL,
                     CLOSED_INTERVAL,
```

```
                    GT_SEMI_INTERVAL,
                    GE_SEMI_INTERVAL,
                    LT_SEMI_INTERVAL,
                    LE_SEMI_INTERVAL,
                    UNBOUNDED )
```

OPEN_INTERVAL denotes the bounded open interval (a, b).

GE_LT_INTERVAL denotes the bounded interval [a, b).

GT_LE_INTERVAL denotes the bounded interval (a, b].

CLOSED_INTERVAL denotes the bounded interval [a, b].

GT_SEMI_INTERVAL denotes the unbounded interval (a, +∞).

GE_SEMI_INTERVAL denotes the unbounded interval [a, +∞).

LT_SEMI_INTERVAL denotes the unbounded interval (-∞, b).

LE_SEMI_INTERVAL denotes the unbounded interval (-∞, b].

UNBOUNDED denotes all values (-∞, +∞).

For angular values, the terms "+∞" and "-∞" denote the most extreme valid values for the coordinate-component.

EXAMPLE 1 In the latitude coordinate-component interval of type GE_SEMI_INTERVAL with value [0.0, +∞), "+∞" denotes +π/2 radians.

EXAMPLE 2 In the longitude coordinate-component interval of type LT_SEMI_INTERVAL with value (-∞, 0.0), "-∞" denotes -π radians.

11.2.6.5 Polar_Aspect

This data type represents the values of the polar aspect parameter of SRFT POLAR_STEREOGRAPHIC.

```
Polar_Aspect ::= (   NORTH,
                     SOUTH )
```

11.2.6.6 SRF_Region_Status

This data type represents coordinate location with respect to the SRF region and/or extended SRF region (see 8.3.2.4) of an SRF.

```
SRF_Region_Status ::= (   IN_SRF_REGION,
                          IN_EXTENDED_REGION_OUTSIDE_SRF_REGION,
                          OUTSIDE_BOTH_SRF_REGION_AND_EXTENDED_REGION )
```

IN_SRF_REGION denotes a coordinate that is contained within the SRF region.

IN_EXTENDED_REGION_OUTSIDE_SRF_REGION denotes a coordinate that is contained within the extended SRF region, but is not contained within the SRF region.

OUTSIDE_BOTH_SRF_REGION_AND_EXTENDED_REGION denotes a coordinate that is contained within the CS domain, but is not contained within either the SRF region or the extended SRF region.

11.2.6.7 SRF_Region_Type

This data type is used to indicate whether an SRF region or extended SRF region is represented in terms of the coordinate system of the SRF or in terms of the geodetic coordinate system of the Celestiodetic SRF for that ORM.

```
SRF_Region_Type ::= (   COORDINATE_REGION,
                        GEODETIC_REGION )
```

`COORDINATE_REGION` denotes an SRF region or extended SRF region that is represented in terms of the coordinate system of the SRF.

`GEODETIC_REGION` denotes an SRF region or extended SRF region that is represented in terms of the geodetic coordinate system of the Celestiodetic SRF for that ORM.

11.2.7 Selection data types

11.2.7.1 Introduction

Selection data types are similar to enumerated data types but form a set of entries that may be extended through registration. Selection data types are defined to be distinct sub-data types of the numeric data type `Integer`, but with specific meanings attached to each value. Selection data types are otherwise processed in the same manner as enumerated data types. The integer codes are unique within each selection data type, but not between data types.

In each selection data type the valid values are 0 and greater. Negative code values are implementation dependent and non-conforming. In each selection data type, the value 0 (`UNSPECIFIED`) is reserved. Some API methods and functions allow 0 (`UNSPECIFIED`) as an input code value and/or an output code value. The valid use of 0 (`UNSPECIFIED`) is defined in the specification of the appropriate method or function.

11.2.7.2 CS_Code

The selection data type `CS_Code` specifies a CS by its code as defined in Clause 5 or by registration. Table 5.7 is a directory of CS specifications, each of which includes a code value and a corresponding label.

11.2.7.3 DSS_Code

The selection data type `DSS_Code` specifies a DSS by its code as defined in Table 9.2 and in Table J.20 or by registration. Each DSS specification includes a code value and a corresponding label.

11.2.7.4 ORM_Code

The selection data type `ORM_Code` specifies an ORM by its code as defined in Annex E and Annex J or by registration. Each ORM specification includes a code value and a corresponding label (see Clause 7).

11.2.7.5 ORMT_Code

The selection data type `ORMT_Code` specifies an ORM Template code defined in Clause 7 or by registration. Table 7.30 is a directory of ORMT specifications, each of which includes a code value and a corresponding label.

11.2.7.6 Profile_Code

The selection data type `Profile_Code` specifies a profile of the SRM by its code as defined in Clause 12 or by registration. Each profile specification includes a code value and a corresponding label.

11.2.7.7 RT_Code

The selection data type `RT_Code` specifies a reference transformation $\boldsymbol{H}_{\mathbf{SR}}$. Each `RT_Code` is specified in Annex E in the entry for the ORM or by registration, specified by the `ORM_Code` value, with which it is associated. Each reference transformation specification associated with an ORM includes a code value and a corresponding label. An `RT_Code` is valid for an ORM only if it has been specified for that ORM. Some API methods also allow the `RT_Code` value `0 (UNSPECIFIED)` to be used.

API methods or functions that require the `RT_Code` data type shall also require its associated `ORM_Code`.

11.2.7.8 SRF_Code

The selection data type `SRF_Code` specifies an SRF by its code as defined in Table 8.31 or by registration. Each SRF specification includes a code value and a corresponding label (see Clause 8).

11.2.7.9 SRFS_Code

The selection data type `SRFS_Code` specifies an SRF set by its code as defined in Table 8.48 or by registration. Each SRF set specification includes a code value and a corresponding label (see Clause 8).

11.2.7.10 SRFS member types

11.2.7.10.1 Introduction

The selection data types that specify the SRF set members associated with the SRF set are defined in Table 8.48.

11.2.7.10.2 Alabama_SPCS_Code

The selection data type `Alabama_SPCS_Code` specifies a member of the Alabama SPCS SRF set in Table 8.50 or by registration.

11.2.7.10.3 GTRS_Global_Coordinate_System_Code

The selection data type `GTRS_Global_Coordinate_System_Code` specifies a member of the GTRS Global Coordinate System SRF set in Table 8.52 and Table 8.53 or by registration.

11.2.7.10.4 Japan_Rectangular_Plane_CS_Code

The selection data type `Japan_Rectangular_Plane_CS_Code` specifies a member of the Japan Rectangular Plane CS SRF set in Table 8.55 or by registration.

11.2.7.10.5 Lambert_NTF_Code

The selection data type `Lambert_NTF_Code` specifies a member of the Lambert NTF SRF set in Table 8.57 or by registration.

11.2.7.10.6 Universal_Polar_Stereographic_Code

The selection data type `Universal_Polar_Stereographic_Code` specifies a member of the Universal Polar Stereographic SRF set in Table 8.59 or by registration.

11.2.7.10.7 Universal_Transverse_Mercator_Code

The selection data type `Universal_Transverse_Mercator_Code` specifies a member of the Universal Transverse Mercator SRF set in Table 8.61 or by registration.

11.2.7.10.8 Wisconsin_SPCS _Code

The selection data type `Wisconsin_SPCS_Code` specifies a member of the Wisconsin SPCS SRF set in Table 8.63 or by registration.

11.2.7.11 SRFT_Code

The selection data type `SRFT_Code` specifies an SRFT by its code as defined in Clause 8 or by registration. Table 8.3 is a directory of SRFT specifications. Each SRFT specification includes a code value and a corresponding label.

11.2.7.12 Status_Code

The `Status_Code` selection data type specifies the status codes associated with the execution of stand-alone functions or class instance methods specified in this International Standard.

This selection data type may be extended in language binding specifications. Values in the range 19-100 are reserved for the future use of this API, while values greater than 100 are reserved for use by language bindings.

```
Status_Code ::= (  0:   UNSPECIFIED,
                   1:   SUCCESS,
                   2:   INVALID_SRF*,
                   3:   INVALID_SOURCE_SRF,
                   4:   INVALID_COORDINATE,
                   5:   INVALID_SOURCE_COORDINATE,
                   6:   INVALID_TARGET_COORDINATE,
                   7:   INVALID_POINT1_COORDINATE,
                   8:   INVALID_POINT2_COORDINATE,
                   9:   OPERATION_UNSUPPORTED,
                   10:  OPERATION_NOT_IMPLEMENTED,
                   11:  INVALID_DIRECTION,
                   12:  INVALID_SOURCE_DIRECTION,
                   13:  INVALID_TARGET_DIRECTION,
                   14:  INVALID_PRIMARY_AXIS_DIRECTION,
                   15:  INVALID_SECONDARY_AXIS_DIRECTION,
                   16:  INVALID_ORIENTATION,
                   17:  INVALID_VECTOR,
                   18:  INVALID_PARAMETERS,
                   19:  INVALID_CODE,
                   20:  UNDEFINED_CODE,
                   21:  INCOMPATIBLE_CODE,
                   22:  INVALID_INPUT,
                   23:  INCOMPATIBLE_INPUTS,
                   24:  DESTRUCTION_FAILURE,
                   25:  FLOATING_OVERFLOW*,
                   26:  FLOATING_UNDERFLOW*,
                   27:  FLOATING_POINT_ERROR*,
                   28:  MEMORY_ALLOCATION_ERROR*,
                   29:  OTHER_RUNTIME_ERROR* )
```

The values marked with an asterisk (“*”) above refer to *common error conditions* that can occur during the execution of most functions and methods, and therefore they are not included in the "Error conditions" element of individual functions or methods. The meanings of these status codes is fully described below, while the meanings of the remaining status codes may vary according to the function or method being specified (see 11.3.2).

`SUCCESS` indicates that the function or method was successfully executed.

`INVALID_SRF` indicates that the SRF instance invoking the method was not successfully created by the API or is otherwise not a valid SRF instance. This condition does not apply to create methods.

`INVALID_SOURCE_SRF` indicates that the SRF instance passed to the method was not successfully created by the API or is otherwise not a valid SRF instance.

`INVALID_COORDINATE` indicates that the coordinate passed to the function or method was either not associated with the specified SRF, or was not within the accuracy domain of the specified SRF.

`INVALID_SOURCE_COORDINATE` indicates that the coordinate passed to the function or method was either not associated with the source SRF, or was not within the accuracy domain of the source SRF.

`INVALID_TARGET_COORDINATE` indicates that the coordinate passed to or returned by the function or method was either not associated with the target SRF, or was not within the accuracy domain of the target SRF.

`INVALID_POINT1_COORDINATE` indicates that the first coordinate passed to the function or method was either not associated with the specified SRF, or was not within the accuracy domain of the specified SRF.

`INVALID_POINT2_COORDINATE` indicates that the second coordinate passed to the function or method was either not associated with the specified SRF, or was not within the accuracy domain of the specified SRF.

`OPERATION_UNSUPPORTED` indicates that the operation associated with the function or method cannot be performed using the specified input parameters.

`OPERATION_NOT_IMPLEMENTED` indicates that the operation associated with the function or method has not yet been implemented.

`INVALID_DIRECTION` indicates that the direction passed to the function or method was either not a valid `Direction` instance, or that its reference coordinate was not associated with the specified SRF, or that its reference coordinate was not within the accuracy domain of the specified SRF.

`INVALID_SOURCE_DIRECTION` indicates that the direction passed to the function or method was either not a valid `Direction` instance, or that its reference coordinate was not associated with the source SRF, or that its reference coordinate was not within the accuracy domain of the source SRF.

`INVALID_TARGET_DIRECTION` indicates that the direction passed to or returned by the function or method was either not associated with the target SRF, or that its reference coordinate was not within the accuracy domain of the target SRF.

`INVALID_PRIMARY_AXIS_DIRECTION` indicates that the primary axis direction passed to the function or method was either not associated with the specified SRF, or that its reference coordinate was not within the accuracy domain of the specified SRF.

`INVALID_SECONDARY_AXIS_DIRECTION` indicates that the secondary axis direction passed to the function or method was either not associated with the specified SRF, or that its reference coordinate was not within the accuracy domain of the specified SRF.

`INVALID_ORIENTATION` indicates that an orientation passed to the function or method was not a valid `Orientation` instance.

`INVALID_VECTOR` indicates that a vector passed to the function or method was not a valid `Vector_3D` data structure.

`INVALID_PARAMETERS` indicates that a parameter data structure is not a valid input to the function or method.

`INVALID_CODE` indicates that an input code value is not a valid input to the function or method.

`UNDEFINED_CODE` indicates that an input code value passed to the function or method is either not defined by this International Standard or is not defined by the implementation.

`INCOMPATIBLE_CODE` indicates that an input code value passed to the function or method is not compatible with the other inputs to that function or method.

`INVALID_INPUT` indicates that an input parameter value cannot be used by the function or method.

`INCOMPATIBLE_INPUTS` indicates that two or more input values passed to the function or method are not compatible with one another.

`DESTRUCTION_FAILURE` indicates that an error occurred during the destruction of an object instance.

`FLOATING_OVERFLOW` indicates that a floating-point overflow error occurred during the execution of the function or method.

`FLOATING_UNDERFLOW` indicates that a floating-point underflow error occurred during the execution of the function or method.

`FLOATING_POINT_ERROR` indicates that a `Long_Float` input is positive or negative infinity, or a not-a-number (NAN) value, or that a floating-point error (other than an overflow or underflow error) occurred during the execution of the function or method.

`MEMORY_ALLOCATION_ERROR` indicates that a memory allocation error occurred during the execution of the function or method.

`OTHER_RUNTIME_ERROR` indicates that some other kind of runtime error occurred during the execution of the function or method.

11.2.7.13 STT_Code

The selection data type `STT_Code` specifies an STT by its code as defined in 7.3.3 or by registration. Each STT specification includes a code value and a corresponding label.

11.2.8 Array data types

11.2.8.1 Introduction

Array data types specify an ordered set whose elements are all of the same data type. Table 11.4 specifies the notation for Array data types.

Table 11.4 — Array data type notation

Data type	Notation
One-dimensional array	Data_Type_Name[length]
Two-dimensional array	Data_Type_Name[rows, columns]

The symbols “length”, "rows", and "columns" are positive integers. The length of a one-dimensional array is specified by "length". The index of the first element in the one-dimensional array is either “0” or “1” depending on the language binding. For two-dimensional arrays, "rows" and "columns" specify the number of rows and columns of the array respectively. The ordering of the set is row-major. The indices of the first element in the two-dimensional array are both either “0” or “1” depending on the language binding.

11.2.8.2 Coordinate2D_Array

This data type specifies an `Object_Reference` array referencing `Coordinate2D` instances. The length of the array is given by the record element `length`.

```
Coordinate2D_Array ::= {
   length                  Integer_Positive;
   coordinate2D_array      Object_Reference[ length ];
}
```

11.2.8.3 Coordinate3D_Array

This data type specifies an `Object_Reference` array referencing `Coordinate3D` instances. The length of the array is given by the record element `length`.

```
Coordinate3D_Array ::= {
   length                  Integer_Positive;
   coordinate3D_array      Object_Reference[ length ];
}
```

11.2.8.4 Direction_Array

This data type specifies an `Object_Reference` array referencing `Direction` instances. The length of the array is given by the record element `length`.

```
Direction_Array ::= {
   length                Integer_Positive;
   direction_array       Object_Reference[ length ];
}
```

11.2.8.5 DSS_Code_Array

This data type specifies an array containing `DSS_Code` values. The length of the array is given by the record element `length`.

```
DSS_Code_Array ::= {
   length                Integer_Unsigned;
   dss_code_array        DSS_Code [ length ];
}
```

11.2.8.6 Matrix_2x2

This data type specifies a two-dimensional square array of four `Long_Float` variables representing a 2x2 matrix.

```
Matrix_2x2 ::= Long_Float[ 2, 2 ]
```

11.2.8.7 Matrix_3x3

This data type specifies a two-dimensional square array of nine `Long_Float` variables representing a 3x3 matrix.

```
Matrix_3x3 ::= Long_Float[ 3, 3 ]
```

11.2.8.8 Matrix_4x4

This data type specifies a two-dimensional square array of sixteen `Long_Float` variables representing a 4x4 matrix.

```
Matrix_4x4 ::= Long_Float[ 4, 4 ]
```

11.2.8.9 ORM_Code_Array

This data type specifies an array containing `ORM_Code` values. The length of the array is given by the record element `length`.

```
ORM_Code_Array ::= {
   length               Integer_Unsigned;
   orm_code_array       ORM_Code [ length ];
}
```

11.2.8.10 Profile_Code_Array

This data type specifies an array containing `Profile_Code` values. The length of the array is given by the record element `length`.

```
Profile_Code_Array ::= {
   length               Integer_Unsigned;
   profile_code_array   Profile_Code [ length ];
}
```

11.2.8.11 RT_Code_Array

This data type specifies an array containing `RT_Code` values. The length of the array is given by the record element `length`.

```
RT_Code_Array ::= {
   length               Integer_Unsigned;
   rt_code_array        RT_Code [ length ];
}
```

11.2.8.12 SRF_Code_Array

This data type specifies an array containing `SRF_Code` values. The length of the array is given by the record element `length`.

```
SRF_Code_Array ::= {
   length               Integer_Unsigned;
   srf_code_array       SRF_Code [ length ];
}
```

11.2.8.13 SRF_Region_Status_Array

This data type specifies an array containing `SRF_Region_Status` values. The length of the array is given by the record element `length`.

```
SRF_Region_Status_Array ::= {
   length               Integer_Positive;
   status_array         SRF_Region_Status[ length ];
}
```

11.2.8.14 SRFS_Code_Array

This data type specifies an array containing `SRFS_Code` values. The length of the array is given by the record element `length`.

```
SRFS_Code_Array ::= {
   length               Integer_Unsigned;
   srfs_code_array      SRFS_Code [ length ];
}
```

11.2.8.15 SRFT_Code_Array

This data type specifies an array containing `SRFT_Code` values. The length of the array is given by the record element `length`.

```
SRFT_Code_Array ::= {
   length               Integer_Unsigned;
   srft_code_array      SRFT_Code [ length ];
}
```

11.2.8.16 Vector_3D

This data type specifies an array of three `Long_Float` variables representing a vector in 3D Euclidean space.

```
Vector_3D ::= Long_Float[ 3 ]
```

11.2.9 Record data types

11.2.9.1 Introduction

Data types that encompass a variety of information are termed *record data types*. A record data type consists of a sequence of data types that together form one record of data. Each entry of a record data type may be of any data type defined in this API, including other record data types.

The elements of record data types that represent lengths shall be evaluated as metres, and the elements that represent angles shall be evaluated as radians.

The following notation is used for defining non-variant record data types:

<Non-Variant Record_Data_Type> ::=
{
  <Variable_Name> <Variable_Data_Type>
  <Variable_Name> <Variable_Data_Type>
  …
}

where:

<Non-Variant_Record_Data_Type>: The non-variant record data type that is being defined.
<Variable_Name>: The name of a record element.
<Variable_Data_Type>: The data type of a record element. Data type “<empty>” indicates no data is present for the element and the data type is left to the language binding.
{}: The body of the non-variant record.

The following notation is used for defining the variant record data types:

<Variant_Record_Data_Type> ::= (<Selector_Name> <Selection_Data_Type>)
{
  <Variable_Name> <Variable_Data_Type>
  <Variable_Name> <Variable_Data_Type>
  …
  [
    <Selection_Name> : <Variable_Name> <Variable_Data_Type>;
    <Selection_Name> : <Variable_Name> <Variable_Data_Type>;
     …
  ]
}

where:

<Selector_Name>: The name of the selector
<Selection_Data_Type>: The selection data type used to select the content of the variant record.
<Variable_Name>: The name of a record element.
<Variable_Data_Type>: The data type of a record element. Data type “<empty>” indicates no data is present for the element and the data type is left to the language binding.
<Selection_Name>: A selection data type value for which a record element applies.
{}: The body of the variant record.
[]: The variant part of the variant record that shall follow all non-varying record elements.

11.2.9.2 Interval

This record data type specifies a coordinate-component interval, consisting of an interval type, a lower bound, and an upper bound. For non-angular intervals, the value of `lower_bound` shall be less than the value of `upper_bound`. For angular intervals, the absolute value of the difference between the bounds shall be less than or equal to 2π. For angular intervals, if the value of `lower_bound` is greater than the value of `upper_bound`, the effective interval is understood to span from the specified value of `lower_bound` to the specified value of `upper_bound` plus 2π.

The value of `lower_bound` is ignored if `interval_type` is a semi-interval `LT_SEMI_INTERVAL`, or `LE_SEMI_INTERVAL`, or `UNBOUNDED`.

The value of `upper_bound` is ignored if `interval_type` is a semi-interval `GT_SEMI_INTERVAL`, or `GE_SEMI_INTERVAL`, or `UNBOUNDED`.

```
Interval ::= {
   interval_type        Interval_Type;
   lower_bound          Long_Float;
   upper_bound          Long_Float;
}
```

11.2.9.3 ORM_Transformation_Parameters

This variant record data type represents a set of 2D or 3D ORM transformation parameters.

```
ORM_Transformation_Parameters ::= { template_code STT_Code }
{
  [
  IDENTITY:
    identity_parameters    <empty>;
  IDENTITY_2D:
    identity_2d_parameters <empty>;
  TRANSLATE:
    translate_parameters                 Translate_3D_Parameters;
  TRANSLATE_2D:
    translate_2d_parameters              Translate_2D_Parameters;
  PV_7_PARAMETER:
    pv_7_parameters                      Rotate_Scale_Translate_3D_Parameters;
  CF_7_PARAMETER:
    cf_7_parameters                      Rotate_Scale_Translate_3D_Parameters;
  CF_7_PLUS_3_PARAMETER:
    cf_7_plus_3_parameters               Molodensky_Badekas_3D_Parameters;
  ROTATE_SCALE_TRANSLATE:
    rotate_scale_translate_parameters    Similarity_3D_Parameters;
  ROTATE_SCALE_TRANSLATE_2D:
    rotate_scale_translate_2d_parameters Similarity_2D_Parameters;
  HOMOGENEOUS_MATRIX_4X4:
    homogeneous_matrix_4x4_parameters    Homogeneous_3D_Parameters;
  HOMOGENEOUS_MATRIX_3X3_2D:
    homogeneous_matrix_3x3_2d_parameters Homogeneous_2D_Parameters;
  CF_XYZ_ROTATE_SCALE_TRANSLATE:
    cf_xyz_rotate_scale_translate_parameters
                                         Rotate_Scale_Translate_3D_Parameters;
  PV_XYZ_ROTATE_TRANSLATE:
    pv_xyz_rotate_translate_parameters   Rotate_Translate_3D_Parameters;
  PV_Z_ROTATE_TRANSLATE:
    pv_z_rotate_translate_parameters     Z_Rotate_Translate_3D_Parameters;
  CF_Z_ROTATE:
    cf_z_rotate_parameters               Z_Rotate_3D_Parameters;
  PV_YZ_ROTATE:
    pv_yz_rotate_parameters              YZ_Rotate_3D_Parameters;
  CF_XZ_ROTATE:
    cf_xz_rotate_parameters              XZ_Rotate_3D_Parameters;
  ]
}
```

11.2.9.4 Orientation representation parameters

11.2.9.4.1 Axis_Angle_Parameters

This record data type specifies the orientation parameters specified in 6.7.3. The rotation angle is given in radians.

```
Axis_Angle_Parameters ::= {
   axis     Vector_3D;
   angle    Long_Float;
}
```

The value of `axis` represents the axis of rotation as a unit vector. The value of `angle` represents the rotation angle in radians. The valid range for values of `angle` is (-2π, 2π).

11.2.9.4.2 Euler_Angles_ZXZ_Parameters

This record data type specifies the orientation parameters specified in 6.7.4.3. It consists of three rotation angles in radians.

```
Euler_Angles_ZXZ_Parameters ::= {
   spin           Long_Float;
   nutation       Long_Float;
   precession     Long_Float;
}
```

The values of `spin`, `nutation`, and `precession` represent consecutive principal rotations in radians about the z–axis, the x–axis and the z–axis again. The valid range for values of `spin`, `nutation`, and `precession` is (-2π, 2π).

11.2.9.4.3 Quaternion_Parameters

This record data type specifies the orientation parameters specified in 6.7.5.1. It consists of a 4-tuple of numbers, the scalar part and the three vector parts. The parameter values shall meet the constraint: $e_0^2 + e_1^2 + e_2^2 + e_3^2 = 1$.

```
Quaternion_Parameters ::= {
   e0       Long_Float;
   e1       Long_Float;
   e2       Long_Float;
   e3       Long_Float;
}
```

11.2.9.4.4 Tait_Bryan_Angles_Parameters

This record data type specifies the orientation parameters specified in 6.7.4.4. It consists of three rotation angles in radians.

```
Tait_Bryan_Angles_Parameters ::= {
   roll      Long_Float;
   pitch     Long_Float;
   yaw       Long_Float;
}
```

The values of `roll`, `pitch`, and `yaw` represent consecutive principal rotations in radians about the x–axis, the y–axis and the z–axis. The valid range for values of `roll`, `pitch`, and `yaw` is (-2π, 2π).

11.2.9.5 SRFT parameter record data types

11.2.9.5.1 EC_Parameters

This record data type specifies the parameters that correspond to SRFT EQUIDISTANT_CYLINDRICAL.

```
EC_Parameters ::= {
   origin_longitude     Long_Float;
   central_scale        Long_Float;
   false_easting        Long_Float;
   false_northing       Long_Float;
}
```

11.2.9.5.2 LCC_Parameters

This record data type specifies the parameters that correspond to SRFT LAMBERT_CONFORMAL_CONIC.

```
LCC_Parameters ::= {
   origin_longitude     Long_Float;
   origin_latitude      Long_Float;
   latitude1            Long_Float;
   latitude2            Long_Float;
   false_easting        Long_Float;
   false_northing       Long_Float;
}
```

11.2.9.5.3 LCE_3D_Parameters

This record data type specifies the parameters that correspond to SRFT LOCOCENTRIC_EUCLIDEAN_3D.

```
LCE_3D_Parameters ::= {
   lococentre           Vector_3D;
   primary_axis         Vector_3D;
   secondary_axis       Vector_3D;
}
```

11.2.9.5.4 LSR_2D_Parameters

This record data type specifies the parameters that correspond to SRFT LOCAL_SPACE_RECTANGULAR_2D.

```
LSR_2D_Parameters ::= {
   forward_direction    Axis_Direction_2D;
}
```

11.2.9.5.5 LSR_3D_Parameters

This record data type specifies the parameters that correspond to SRFT LOCAL_SPACE_RECTANGULAR_3D.

```
LSR_3D_Parameters ::= {
   forward_direction    Axis_Direction_3D;
   up_direction         Axis_Direction_3D;
}
```

11.2.9.5.6 Local_Tangent_Parameters

This record data type specifies the parameters that correspond to SRFT LOCAL_TANGENT_SPACE_AZIMUTHAL_SPHERICAL, and SRFT LOCAL_TANGENT_SPACE_CYLINDRICAL.

```
Local_Tangent_Parameters ::={
   geodetic_longitude   Long_Float;
   geodetic_latitude    Long_Float;
   azimuth              Long_Float;
   height_offset        Long_Float;
}
```

11.2.9.5.7 LTSE_Parameters

This record data type specifies the parameters that correspond to SRFT LOCAL_TANGENT_SPACE_EUCLIDEAN.

```
LTSE_Parameters ::={
   geodetic_longitude   Long_Float;
   geodetic_latitude    Long_Float;
   azimuth              Long_Float;
   x_false_origin       Long_Float;
   y_false_origin       Long_Float;
   height_offset        Long_Float;
}
```

11.2.9.5.8 M_Parameters

This record data type specifies the parameters that correspond to SRFT MERCATOR.

```
M_Parameters ::= {
   origin_longitude     Long_Float;
   central_scale        Long_Float;
   false_easting        Long_Float;
   false_northing       Long_Float;
}
```

11.2.9.5.9 Oblique_Mercator_Parameters

This record data type specifies the parameters that correspond to SRFT OBLIQUE_MERCATOR_SPHERICAL.

```
Oblique_Mercator_Parameters ::= {
   longitude1           Long_Float;
   latitude1            Long_Float;
   longitude2           Long_Float;
   latitude2            Long_Float;
   central_scale        Long_Float;
   false_easting        Long_Float;
   false_northing       Long_Float;
}
```

11.2.9.5.10 PS_Parameters

This record data type specifies the parameters that correspond to SRFT POLAR_STEREOGRAPHIC.

```
PS_Parameters ::= {
   polar_aspect          Polar_Aspect;
   origin_longitude      Long_Float;
   central_scale         Long_Float;
   false_easting         Long_Float;
   alse_northing         Long_Float;
}
```

11.2.9.5.11 SRFS_Code_Info

This variant record data type specifies an arbitrary `SRFS_Code` with its associated SRF set member code.

```
SRFS_Code_Info ::= ( srfs_code    SRFS_Code )
{
  [
    UNSPECIFIED:
      unspecified                       <empty>;
    ALABAMA_SPCS:
      alabama_spcs                      Alabama_SPCS_Code;
    GTRS_GLOBAL_COORDINATE_SYSTEM:
      gtrs_global_coordinate_system     GTRS_Global_Coordinate_System_Code;
    JAPAN_RECTANGULAR_PLANE_CS:
      japan_rectangular_plane_cs        Japan_Rectangular_Plane_CS_Code;
    LAMBERT_NTF:
      lambert_ntf                       Lambert_NTF_Code;
    UNIVERSAL_POLAR_STEREOGRAPHIC:
      universal_polar_stereographic     Universal_Polar_Stereographic_Code;
    UNIVERSAL_TRANSVERSE_MERCATOR:
      universal_transverse_mercator     Universal_Transverse_Mercator_Code;
    WISCONSIN_SPCS:
      wisconsin_spcs                    Wisconsin_SPCS_Code;
  ]
}
```

11.2.9.5.12 TM_Parameters

This record data type specifies the parameters that correspond to SRFT TRANSVERSE_MERCATOR.

```
TM_Parameters ::= {
   origin_longitude      Long_Float;
   origin_latitude       Long_Float;
   central_scale         Long_Float;
   false_easting         Long_Float;
   false_northing        Long_Float;
}
```

11.2.9.6 STT parameter record data types

11.2.9.6.1 Homogeneous_2D_Parameters

This record data type represents the parameters of a 2D homogeneous transformation, consisting of the three origin displacement components and a scaled rotation submatrix, as specified in Table 7.22.

```
Homogeneous_2D_Parameters ::= {
   delta_x               Long_Float;
```

```
   delta_y              Long_Float;
   scaled_rotation      Matrix_2x2;
}
```

The values of `delta_x` and `delta_y` represent the origin displacement in metres.

The value of `scaled_rotation` represents the scaled rotation submatrix. Therefore, its determinant shall be greater than zero, and its inverse shall equal its transpose divided by the square of its determinant.

11.2.9.6.2 Homogeneous_3D_Parameters

This record data type represents the parameters of a 3D homogeneous transformation, consisting of the three origin displacement components and a scaled rotation submatrix, as specified in Table 7.21.

```
Homogeneous_3D_Parameters ::= {
   delta_x              Long_Float;
   delta_y              Long_Float;
   delta_z              Long_Float;
   scaled_rotation      Matrix_3x3;
}
```

The values of `delta_x`, `delta_y`, and `delta_z` represent the origin displacement in metres.

The value of `scaled_rotation` represents the scaled rotation submatrix. Therefore, its determinant shall be greater than zero, and its inverse shall equal its transpose divided by the square of its determinant.

11.2.9.6.3 Molodensky_Badekas_3D_Parameters

This record data type represents the parameters of a Molodensky-Badekas 3D transformation as specified in Table 7.18.

```
Molodensky_Badekas_3D_Parameters ::= {
   delta_x         Long_Float;
   delta_y         Long_Float;
   delta_z         Long_Float;
   omega_1         Long_Float;
   omega_2         Long_Float;
   omega_3         Long_Float;
   delta_scale     Long_Float;
   x_0             Long_Float;
   y_0             Long_Float;
   z_0             Long_Float;
}
```

The values of `delta_x`, `delta_y`, and `delta_z` represent the origin displacement in metres.

The values of `omega_1`, `omega_2`, and `omega_3` represent rotations in radians. In general, the valid range for values of `omega_1`, `omega_2`, and `omega_3` is $(-2\pi, 2\pi)$. See the STT specification for additional constraints.

The value of `delta_scale` represents a scale difference from unity; the sum `(1+delta_scale)` represents the scale factor of the transformation. The value of the scale factor shall be greater than zero, therefore the value of `delta_scale` shall be greater than -1. See the STT specification for additional constraints.

The values of `x_0`, `y_0`, and `z_0` represent the initial point in metres.

11.2.9.6.4 Rotate_Scale_Translate_3D_Parameters

This record data type represents the parameters of a general 3D transformation as specified in Table 7.16, Table 7.17, and Table 7.23.

```
Rotate_Scale_Translate_3D_Parameters ::= {
   delta_x         Long_Float;
   delta_y         Long_Float;
   delta_z         Long_Float;
   omega_1         Long_Float;
   omega_2         Long_Float;
   omega_3         Long_Float;
   delta_scale     Long_Float;
}
```

The values of `delta_x`, `delta_y`, and `delta_z` represent the origin displacement in metres.

The values of `omega_1`, `omega_2`, and `omega_3` represent rotations in radians. In general, the valid range for values of `omega_1`, `omega_2`, and `omega_3` is (-2π, 2π). See the applicable STT specifications for additional constraints.

The value of `delta_scale` represents a scale difference from unity; the sum `(1+delta_scale)` represents the scale factor of the transformation. The value of the scale factor shall be greater than zero, therefore the value of `delta_scale` shall be greater than -1. See the applicable STT specification for additional constraints.

11.2.9.6.5 Rotate_Translate_3D_Parameters

This record data type represents the parameters of a 3D translation of the origin and rotation about the axes as specified in Table 7.24.

```
Rotate_Translate_3D_Parameters ::= {
   delta_x         Long_Float;
   delta_y         Long_Float;
   delta_z         Long_Float;
   omega_1         Long_Float;
   omega_2         Long_Float;
   omega_3         Long_Float;
}
```

The values of `delta_x`, `delta_y`, and `delta_z` represent the origin displacement in metres.

The values of `omega_1`, `omega_2`, and `omega_3` represent rotations in radians. In general, the valid range for values of `omega_1`, `omega_2`, and `omega_3` is (-2π, 2π). See the STT specification for additional constraints.

11.2.9.6.6 Similarity_2D_Parameters

This record data type represents the parameters of a 2D general similarity transformation, as specified in Table 7.20.

```
Similarity_2D_Parameters ::= {
   delta_x         Long_Float;
   delta_y         Long_Float;
   rotation        Matrix_2x2;
   scale           Long_Float;
}
```

The values of `delta_x` and `delta_y` represent the origin displacement in metres.

The value of `rotation` represents a rotation matrix. Therefore, its inverse shall equal its transpose, and its determinant shall equal one.

The value of `scale` represents the scale factor of the transformation. The value of `scale` shall be greater than zero.

11.2.9.6.7 Similarity_3D_Parameters

This record data type represents the parameters of a 3D general similarity transformation, as specified in Table 7.19.

```
Similarity_3D_Parameters ::= {
   delta_x         Long_Float;
   delta_y         Long_Float;
   delta_z         Long_Float;
   rotation        Matrix_3x3;
   scale           Long_Float;
}
```

The values of `delta_x`, `delta_y`, and `delta_z` represent the origin displacement in metres.

The value of `rotation` represents a rotation matrix. Therefore, its inverse shall equal its transpose, and its determinant shall equal one.

The value of `scale` represents the scale factor of the transformation. The value of `scale` shall be greater than zero.

11.2.9.6.8 Translate_2D_Parameters

This record data type represents the parameters of a 2D translation of the origin as specified in Table 7.15.

```
Translate_2D_Parameters ::= {
   delta_x         Long_Float;
   delta_y         Long_Float;
}
```

The values of `delta_x` and `delta_y` represent the origin displacement in metres.

11.2.9.6.9 Translate_3D_Parameters

This record data type represents the parameters of a 3D translation of the origin, as specified in Table 7.14.

```
Translate_3D_Parameters ::= {
   delta_x         Long_Float;
   delta_y         Long_Float;
   delta_z         Long_Float;
}
```

The values of `delta_x`, `delta_y`, and `delta_z` represent the origin displacement in metres.

11.2.9.6.10 XZ_Rotate_3D_Parameters

This record data type represents the parameters of a 3D rotation about the ***x***- and ***z***-axes as specified in Table 7.28.

```
XZ_Rotate_3D_Parameters ::= {
   omega_1          Long_Float;
   omega_3          Long_Float;
}
```

The values of `omega_1` and `omega_3` represent rotations in radians about the ***x***- and ***z***-axes. In general, the valid range for values of `omega_1` and `omega_3` is (-2π, 2π). See the STT specification for additional constraints.

11.2.9.6.11 YZ_Rotate_3D_Parameters

This record data type represents the parameters of a 3D rotation about the ***y***- and ***z***-axes as specified in Table 7.27.

```
YZ_Rotate_3D_Parameters ::= {
   omega_2     Long_Float;
   omega_3     Long_Float;
}
```

The values of `omega_2` and `omega_3` represent rotations in radians about the ***y***- and ***z***-axes. In general, the valid range for values of `omega_2` and `omega_3` is (-2π, 2π). See the STT specification for additional constraints.

11.2.9.6.12 Z_Rotate_3D_Parameters

This record data type represents the parameters of a 3D rotation about the ***z***-axis as specified in Table 7.26.

```
Z_Rotate_3D_Parameters ::= {
   omega_3        Long_Float;
}
```

The value of `omega_3` represents a rotation in radians about the ***z***-axis. In general, the valid range for values of `omega_3` is (-2π, 2π). See the STT specification for additional constraints.

11.2.9.6.13 Z_Rotate_Translate_3D_Parameters

This record data type represents the parameters of a 3D translation of the origin and rotation about the ***z***-axis as specified in Table 7.25.

```
Z_Rotate_Translate_3D_Parameters ::= {
   delta_x        Long_Float;
   delta_y        Long_Float;
   delta_z        Long_Float;
   omega_3        Long_Float;
}
```

The values of `delta_x`, `delta_y`, and `delta_z` represent the origin displacement in metres.

The value of `omega_3` represents a rotation in radians about the ***z***-axis. In general, the valid range for values of `omega_3` is (-2π, 2π). See the STT specification for additional constraints.

11.3 Object classes

11.3.1 Introduction

SRF classes specify methods that implement the spatial operations specified in Clause 10. To aid in specification, most of the functionality of the API is defined using a class hierarchy with each abstract class providing the specification of those methods that are common to each of its subclasses. The class inheritance hierarchy is summarized by the UML class diagram shown in Figure 11.1, in which arrows indicate the parent class of each subclass. The remaining functionality is provided in concrete class and method specifications. Only concrete classes can be instanced. The implementation of abstract classes is not required.

The functionality of the methods are specified in the class specification tables (see 11.3.2) that provide the method name, the semantics, inputs and outputs of the method, and the error conditions of the method. These class specifications use phrases such as "this SRF" to refer to the internal state of an instance of the class.

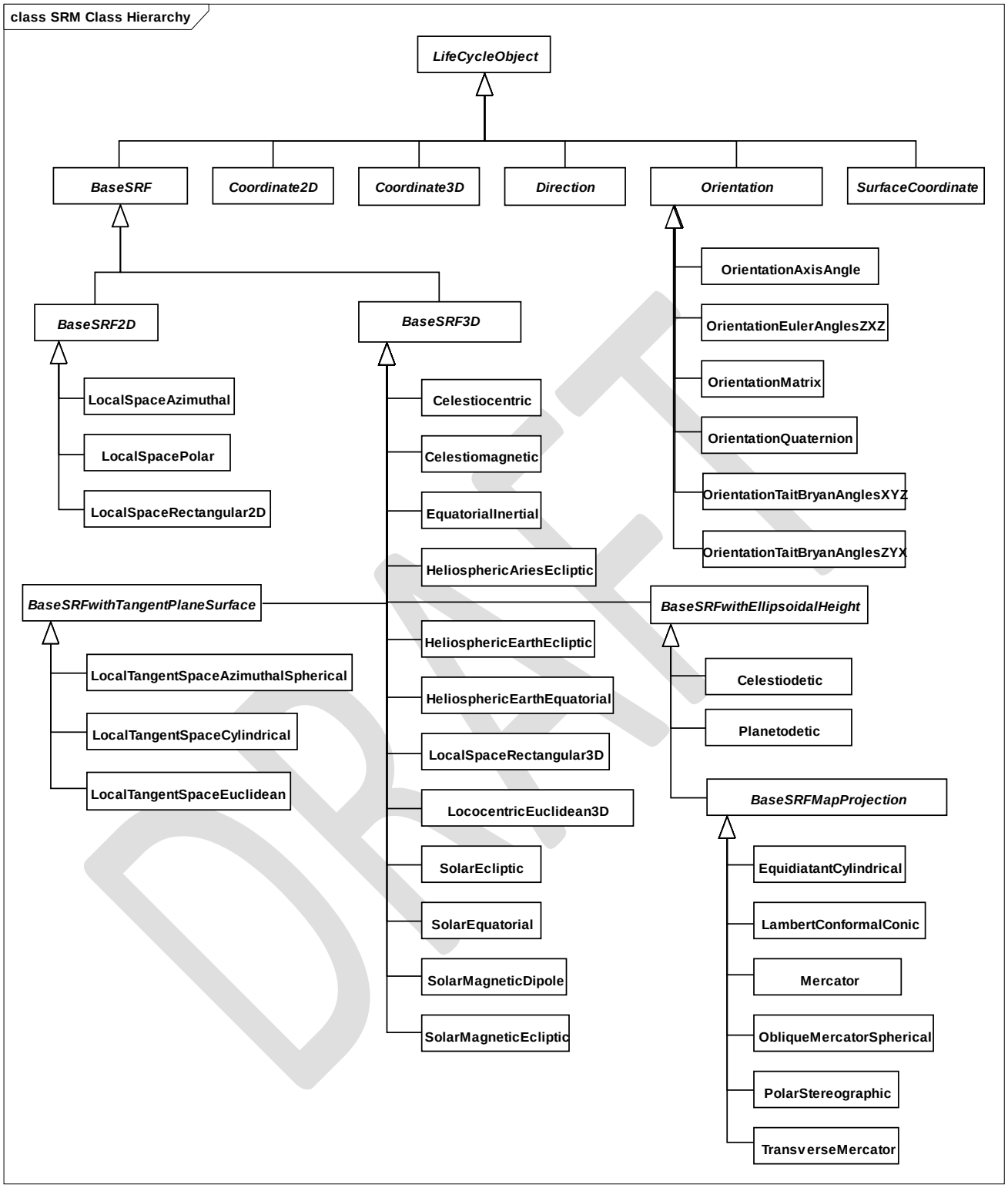

Figure 11.1 — Class inheritance hierarchy summary

11.3.2 Class specification format

Class data types are specified in tables in Table 11.5 through Table 11.66 with the following elements:

Table 11.5 — Class specification elements

Element	Definition
Class	The name of the object class.
Description	The corresponding SRM concept.
Superclass(es)	The specification of inherited functionality listing the superclasses of the class in hierarchical order. Each superclass name is followed by a list of the methods it specifies. The method list excludes methods that are overridden.
Method or Abstract method	The name of the method.
Semantics	The specification of the method functionality.
Inputs	The specification(s) of the method input parameters, or "none". The state of the invoking class instance is implicitly an input and is not additionally listed in this element. The Create method of a class is an exception since it depends only on its explicit input parameters.
Outputs	The specification(s) of the method output parameters, or “none”.
Error conditions	The list of `Status_Code` values that correspond to error conditions. Each listed value specifies the condition for which it is applicable. Common error conditions (see 11.2.7.12) are not listed in this element. The success condition (see 11.2.7.12) is not listed in this element. Unless otherwise specified, all outputs from a method are undefined when an error condition is encountered.

The method element of a concrete class is labelled “Method”. The method element of an abstract class is labelled “Abstract method”. A subclass inherits all the abstract methods of its superclass including methods that the superclass has inherited. In particular, an abstract method inherited by a concrete class shall be implemented for the concrete class. An implementation may implement private methods in concrete classes for internal use, but public access to a concrete implementation of a private method is not a requirement. The order in which the methods are listed in each class follows a life cycle pattern, e.g., creation methods, then data manipulation methods, and then spatial operations.

The success condition is associated with `Status_Code SUCCESS` (see 11.2.7.12). The success condition is a nominal behaviour of all methods and is not listed within the Error conditions element. The error conditions applicable to a method invocation are the common error conditions specified in 11.2.7.12, the additional error conditions specified in the Error conditions element for the method, and any language-binding specific error conditions applicable to the method.

Unless otherwise indicated by the method semantics, output values are undefined under an error condition. When several error conditions apply to a method invocation, the first error condition detected by an implementation shall be presented as the method status.

Language bindings may add additional error conditions and related binding-specific mechanisms including the passing of inputs and outputs, and the presentation of method status. Language bindings shall specify these mechanisms, since this International Standard does not restrict such mechanisms. A language binding mechanism for presentation of method status shall support the association of a unique error `Status_Code` (see 11.2.7.12) for an error condition. If a language binding supports exception handling and if a language binding uses that mechanism to present method failure, then a method or property of the exception that provides the corresponding `Status_Code` would satisfy this requirement.

11.3.3 LifeCycleObject

The `LifeCycleObject` class is the abstract class from which all other classes inherit. All other abstract classes are defined in 11.3.5. Instances of all concrete subclasses of `LifeCycleObject` are created dynamically using the `Create` method. Once created, each instance will continue to exist and respond to method invocations until it is destroyed using the corresponding `Destroy` method. This standard sequence of instance creation, method execution, and destruction is termed the *object life cycle*.

Table 11.6 — LifeCycleObject

Element	Specification
Class	`LifeCycleObject`
Description	The abstract class from which all other classes inherit. An instance of a subclass derived from `LifeCycleObject` is not valid until the `Create` method is successfully invoked. A valid class instance becomes invalid after the `Destroy` method is invoked.
Superclass(es)	none
Abstract method	`Create`
Semantics	Creates an instance of a concrete class. An implementation may perform memory allocation and/or class instance initialization as part of this method.
Inputs	Specific inputs are specified in concrete classes that are directly or indirectly subclassed from this class.
Outputs	`object_reference:` `Object_Reference`
Error conditions	No additional error conditions.
Abstract method	`Destroy`
Semantics	Destroys an instance of a concrete class. An implementation may perform memory deallocation and/or class instance finalization as part of this method.
Inputs	`object_reference:` `Object_Reference`
Outputs	none
Error conditions	`DESTRUCTION_FAILURE` if the class instance was not successfully deallocated.

There are restrictions on the order in which the methods of classes derived from `LifeCycleObject` may be invoked. The restrictions are:

a) The `Create` method of a life cycle object shall be the first method invoked on the class.

b) The `Destroy` method of a life cycle object shall be the last method invoked on any instance of the class. Depending on the language binding and the language capabilities, invocation of the `Destroy` method may be:

   1) explicit – invoked by the API user,
   2) implicit – managed by the runtime system, or
   3) implicit/optionally explicit – managed by the runtime system if not explicitly invoked by the API user on any single instance.

c) All other methods shall only be invoked after the `Create` method and before the `Destroy` method.

The following examples illustrate the method sequence for life cycle objects.

NOTE The status for each invocation of a method or function should be checked for error conditions. For brevity and clarity this status checking is not shown in these examples.

EXAMPLE 1 Find the Euclidean distance between two locations.

```
--Note: Label in italics denotes a symbolic constant for this example --
Celestiodetic method Create(
   Inputs: N_AM_1983, N_AM_1983_CONUS;
   Output: ex1_srf)
ex1_srf method CreateCoordinate3D(
   Inputs: -77°(π/180°), +38°(π/180°), 0;
   Output: coordinate1)
ex1_srf method CreateCoordinate3D(
   Inputs +3°(π/180°), +49°(π/180°), 0;
    Output: coordinate2)
ex1_srf method EuclideanDistance(
   Inputs coordinate1, coordinate2;
   Output: distance)
-- use distance result --
coordinate1 method Destroy
coordinate2 method Destroy
ex1_srf method Destroy
```

EXAMPLE 2 Change SRF representation of a location from UTM to Geocentric

```
--Note: Labels in italics denote symbolic constants for this example --
Function    CreateSRFSetMember(
   Inputs: ( UNIVERSAL_TRANSVERSE_MERCATOR,
             ZONE_23_NORTHERN_HEMISPHERE ),
             N_AM_1983,
             N_AM_1983_CONUS,
   Output: source_srf)
source_srf method CreateCoordinate3D(
   Inputs: 350000, 400, 0,
   Output: source_coordinate)
Function CreateStandardizedSRF(
   Inputs: SRF_GEOCENTRIC_WGS_1984, RT_WGS_1984_IDENTITY,
   Output: target_srf)
target_srf method ChangeCoordinate3DSRF(
Inputs: source_srf, source_coordinate,
Outputs: target_coordinate, region)
-- check region value and use result --
source_coordinate method Destroy
target_coordinate method Destroy
source_srf method Destroy
target_srf method Destroy
```

11.3.4 Coordinate and direction classes

11.3.4.1 Introduction

The coordinate and direction classes specified in this subclause are concrete classes that expose no additional methods. Instances of these classes are used in various methods of the SRF classes specified in 11.3.5 through 11.3.10.

11.3.4.2 Coordinate2D

An instance of the `Coordinate2D` class represents a coordinate of a 2D CS.

Table 11.7 — Coordinate2D

Element	Specification
Class	`Coordinate2D`
Description	A coordinate of a 2D CS (see 5.4).
Superclass(es)	`LifeCycleObject`: `Create`, `Destroy`

11.3.4.3 Coordinate3D

An instance of the `Coordinate3D` class represents a coordinate of a 3D CS.

Table 11.8 — Coordinate3D

Element	Specification
Class	`Coordinate3D`
Description	A coordinate of a 3D CS (see 5.4).
Superclass(es)	`LifeCycleObject`: `Create`, `Destroy`

11.3.4.4 Direction

An instance of the `Direction` class represents a spatial direction including the reference coordinate and a unit vector in the local tangent frame of the SRF at the reference coordinate.

Table 11.9 — Direction

Element	Specification
Class	`Direction`
Description	A direction (see 10.5.3).
Superclass(es)	`LifeCycleObject`: `Create`, `Destroy`

11.3.4.5 SurfaceCoordinate

An instance of the `SurfaceCoordinate` class represents a coordinate of a surface CS.

Table 11.10 — SurfaceCoordinate

Element	Specification
Class	`SurfaceCoordinate`
Description	A coordinate of a surface CS (see 5.4).
Superclass(es)	`LifeCycleObject`: `Create`, `Destroy`

11.3.5 Abstract classes

11.3.5.1 BaseSRF

This is the base class for all SRF classes. `BaseSRF` has two abstract subclasses, `BaseSRF2D` and `BaseSRF3D`, as shown in Figure 11.2. `BaseSRF` provides the following (Table 11.11) methods common to all SRF classes:

`GetCSCode`,
`GetORMCodes`, and
`GetSRFCodes`.

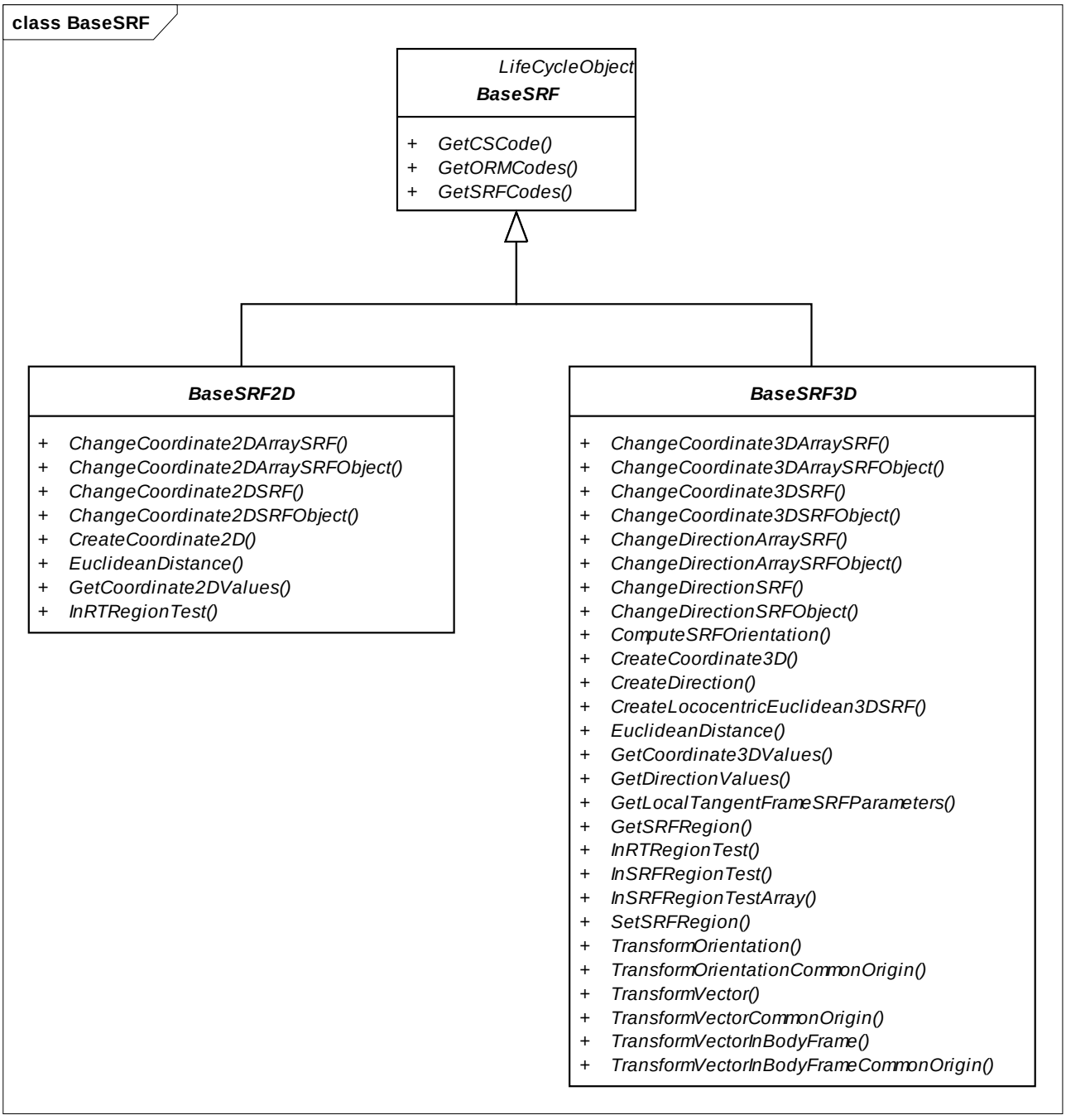

Figure 11.2 — BaseSRF class hierarchy

Table 11.11 — BaseSRF

Element	Specification
Class	BaseSRF
Description	An abstract class specifying the common methods of all SRF classes.
Superclass(es)	LifeCycleObject: Create, Destroy
Abstract method	GetCSCode
Semantics	Outputs the CS_Code code of this SRF.
Inputs	none
Outputs	cs_code: CS_Code
Error conditions	No additional error conditions.
Abstract method	GetORMCodes
Semantics	Outputs the ORM_Code and the RT_Code of this SRF.
Inputs	none
Outputs	orm_code: ORM_Code rt_code: RT_Code
Error conditions	No additional error conditions.
Abstract method	GetSRFCodes
Semantics	1) Outputs the SRFT_Code of this SRF. 2) If created by the CreateStandardizedSRF function, outputs a valid SRF_Code (otherwise outputs 0). (See 11.4.) 3) If created by the CreateSRFSetMember function, outputs a valid SRFS_Code_Info (otherwise outputs the SRFS_Code_Info with SRFS_Code selector value set to UNSPECIFIED). (See 11.5.)
Inputs	none
Outputs	srft_code: SRFT_Code srf_code: SRF_Code srfs_code_info: SRFS_Code_Info
Error conditions	No additional error conditions.

11.3.5.2 BaseSRF2D

This is the base class for all 2D SRF classes. BaseSRF2D is a subclass of BaseSRF. BaseSRF2D has three concrete subclasses, as shown in Figure 11.3. This abstract class adds the following methods, which are specified in Table 11.12:

ChangeCoordinate2DArraySRF,
ChangeCoordinate2DArraySRFObject,
ChangeCoordinate2DSRF,
ChangeCoordinate2DSRFObject,
CreateCoordinate2D,
EuclideanDistance,
GetCoordinate2DValues, and
InRTRegionTest.

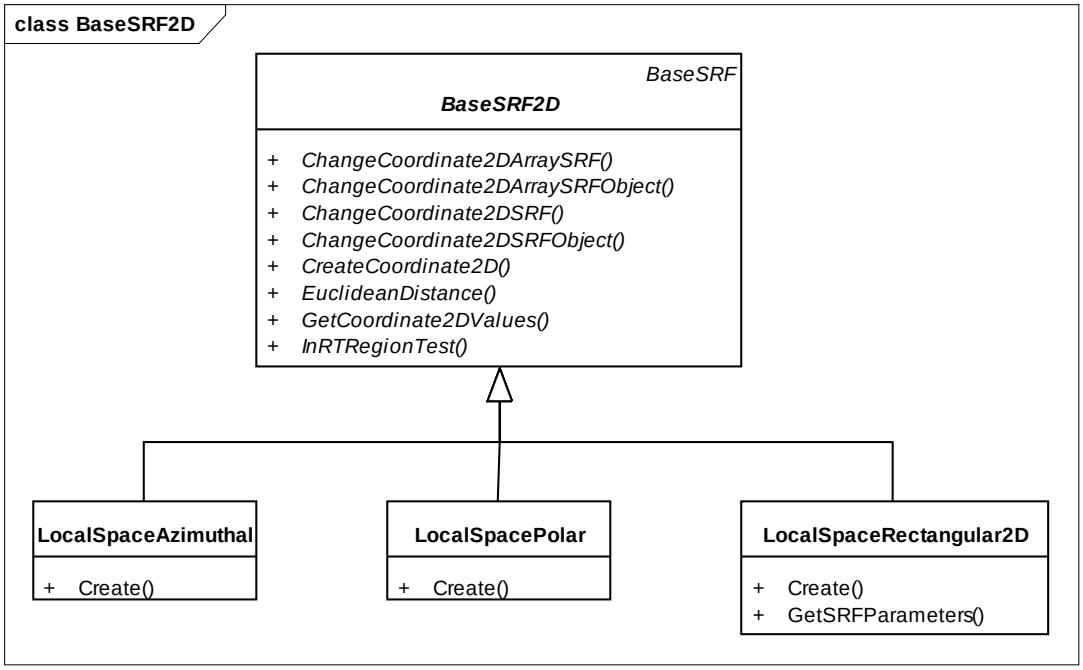

Figure 11.3 — BaseSRF2D class hierarchy

Table 11.12 — BaseSRF2D

Element	Specification
Class	`BaseSRF2D`
Description	An abstract class specifying the common methods for SRF classes with CS of type 2D.
Superclass(es)	`LifeCycleObject`: `Create`, `Destroy` `BaseSRF`: `GetCSCode`, `GetORMCodes`, `GetSRFCodes`
Abstract method	`ChangeCoordinate2DArraySRF`
Semantics	Performs the same operation defined for the `ChangeCoordinate2DSRF` method on each `Coordinate2D` instance in `source_coordinate_array`. The processing is in array indexing order. Upon an error condition, the processing is halted and the output `index` is set to the array index of the offending `Coordinate2D` instance. When successful, the output `index` is set to the size of the array plus one.
Inputs	`source_srf:` `BaseSRF2D` `source_coordinate_array:` `Coordinate2D_Array`
Outputs	`target_coordinate_array:` `Coordinate2D_Array` `index:` `Integer_Unsigned`

Element	Specification
Class	BaseSRF2D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_INPUT if source_coordinate_array is not a valid Coordinate2D_Array data structure. 3) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 4) INVALID_SOURCE_COORDINATE if the Coordinate2D instance at index in source_coordinate_array is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 5) INVALID_TARGET_COORDINATE if the spatial position of the Coordinate2D instance at index is not in the accuracy domain of this SRF.
Abstract method	ChangeCoordinate2DArraySRFObject
Semantics	Performs the same operation defined for the ChangeCoordinate2DSRFObject method on each Coordinate2D instance in source_coordinate_array. The processing is in array indexing order. Upon an error condition, the processing is halted and the output index is set to the array index of the offending Coordinate2D instance. When successful, the output index is set to the size of the array plus one.
Inputs	source_srf: BaseSRF2D source_coordinate_array: Coordinate2D_Array h_st: ORM_Transformation_Parameters
Outputs	target_coordinate_array: Coordinate2D_Array index: Integer_Unsigned
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_INPUT if source_coordinate_array is not a valid Coordinate2D_Array data structure. 3) INVALID_PARAMETERS if the input h_st parameter values are not valid, or are not of the correct dimension. 4) INVALID_SOURCE_COORDINATE if the Coordinate2D instance at index in source_coordinate_array is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 5) INVALID_TARGET_COORDINATE if the spatial position of the Coordinate2D instance at index is not in the accuracy domain of this SRF.
Abstract method	ChangeCoordinate2DSRF
Semantics	Changes the SRF representation of the spatial position specified by the input Coordinate2D source_coordinate in the source SRF source_srf to a Coordinate2D target_coordinate in this SRF, the target SRF, in accordance with 10.4.2 using the implicit ORM transformation $\boldsymbol{H}_{\mathbf{ST}}$ given in Equation (10.2). Equation (10.2) assumes both SRFs are based on object-fixed ORMs for the same spatial object.
Inputs	source_srf: BaseSRF2D source_coordinate: Coordinate2D
Outputs	target_coordinate: Coordinate2D

Element	Specification
Class	BaseSRF2D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_coordinate is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 4) INVALID_TARGET_COORDINATE if the spatial position of source_coordinate is not in the accuracy domain of this SRF.
Abstract method	ChangeCoordinate2DSRFObject
Semantics	Changes the SRF representation of the spatial position specified by the input Coordinate2D source_coordinate in the source SRF source_srf to a Coordinate2D target_coordinate in this SRF, the target SRF, using an explicit ORM transformation $\boldsymbol{H}_{\mathbf{ST}}$ as specified by input h_st in accordance with 10.4.2. The input h_st is required in the case of SRFs for different spatial objects or in the case that one or both ORM reference transformations have not been specified.
Inputs	source_srf: BaseSRF2D source_coordinate: Coordinate2D h_st: ORM_Transformation_Parameters
Outputs	target_coordinate: Coordinate2D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_coordinate is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_PARAMETERS if the input h_st parameter values are not valid, or are not of the correct dimension. 4) INVALID_TARGET_COORDINATE if the spatial position of source_coordinate is not in the accuracy domain of this SRF.
Abstract method	CreateCoordinate2D
Semantics	Creates a Coordinate2D instance associated with this SRF from two ordered coordinate-component values. Coordinate-components that represent lengths shall be evaluated as metres. Coordinate-components that represent angles shall be evaluated as radians.
Inputs	first_coordinate_component: Long_Float second_coordinate_component: Long_Float
Outputs	new_coordinate: Coordinate2D
Error conditions	INVALID_INPUT if the spatial position specified by the input coordinate-component values is not in the accuracy domain of this SRF.
Abstract method	EuclideanDistance
Semantics	Outputs the Euclidean distance in metres between the spatial points represented by Coordinate2D instances point1_coordinate and point2_coordinate (see 10.6).
Inputs	point1_coordinate: Coordinate2D point2_coordinate: Coordinate2D
Outputs	distance: Long_Float

Element	Specification
Class	BaseSRF2D
Error conditions	1) INVALID_POINT1_COORDINATE if point1_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_POINT2_COORDINATE if point2_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	GetCoordinate2DValues
Semantics	Retrieves the two ordered coordinate-components of a Coordinate2D instance. Coordinate-components that represent lengths shall be expressed in metres. Coordinate-components that represent angles shall be expressed in radians.
Inputs	coordinate: Coordinate2D
Outputs	first_coordinate_component: Long_Float second_coordinate_component: Long_Float
Error conditions	INVALID_COORDINATE if coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	InRTRegionTest
Semantics	Determines whether the specified coordinate is within the RT region for this SRF. If the RT region is specified in this International Standard, is_set is returned as true, and in_region is returned as true if coordinate is within the RT region. If the RT region is not specified, is_set is returned as false, and in_region is returned as true.
Inputs	coordinate: Coordinate2D
Outputs	is_set: Boolean in_region: Boolean
Error conditions	INVALID_COORDINATE if coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.

11.3.5.3 BaseSRF3D

This is the base class for all 3D SRF classes. BaseSRF3D is a subclass of BaseSRF. BaseSRF3D has both abstract and concrete subclasses, as shown in Figure 11.4. This abstract class adds the following methods, which are specified in Table 11.13:

ChangeCoordinate3DArraySRF,
ChangeCoordinate3DArraySRFObject,
ChangeCoordinate3DSRF,
ChangeCoordinate3DSRFObject,
ChangeDirectionArraySRF,
ChangeDirectionArraySRFObject,
ChangeDirectionSRF,
ChangeDirectionSRFObject,
ComputeSRFOrientation,
CreateCoordinate3D,
CreateDirection,
CreateLococentricEuclidean3DSRF,
EuclideanDistance,
GetCoordinate3DValues,
GetDirectionValues,
GetLocalTangentFrameSRFParameters,
GetSRFRegion,

`InRTRegionTest`,
`InSRFRegionTest`,
`InSRFRegionTestArray`,
`SetSRFRegion`,
`TransformOrientation`,
`TransformOrientationCommonOrigin`,
`TransformVector`,
`TransformVectorCommonOrigin`,
`TransformVectorInBodyFrame`, and
`TransformVectorInBodyFrameCommonOrigin`.

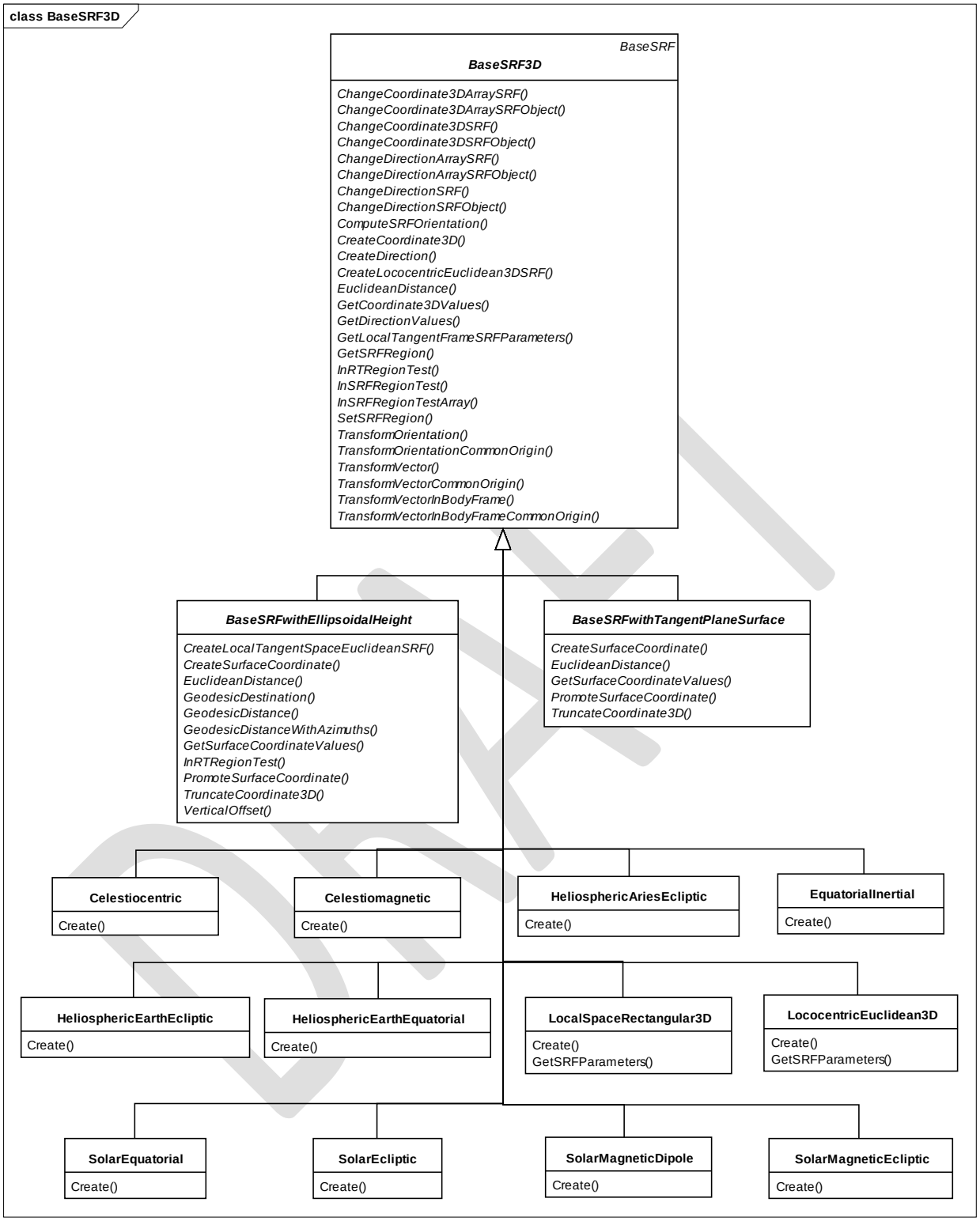

Figure 11.4 — BaseSRF3D class hierarchy

Table 11.13 — BaseSRF3D

Element	Specification
Class	BaseSRF3D
Description	An abstract class specifying the common methods for SRF classes with CS of type 3D.
Superclass(es)	LifeCycleObject: Create, Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes
Abstract method	ChangeCoordinate3DArraySRF
Semantics	Performs the same operation defined for the ChangeCoordinate3DSRF method on each Coordinate3D instance in source_coordinate_array. The processing is in array indexing order. Upon an error condition, the processing is halted and the output index is set to the array index of the offending Coordinate3D instance. When successful, the output index is set to the size of the array plus one.
Inputs	source_srf: BaseSRF3D source_coordinate_array: Coordinate3D_Array
Outputs	target_coordinate_array: Coordinate3D_Array index: Integer_Unsigned
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_INPUT if source_coordinate_array is not a valid Coordinate3D_Array data structure. 3) INVALID_SOURCE_COORDINATE if the Coordinate3D instance at index in source_coordinate_array is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 4) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 5) INVALID_TARGET_COORDINATE if the spatial position of the Coordinate3D instance at index is not in the accuracy domain of this SRF.
Abstract method	ChangeCoordinate3DArraySRFObject
Semantics	Performs the same operation defined for the ChangeCoordinate3DSRFObject method on each Coordinate3D instance in source_coordinate_array. The processing is in array indexing order. Upon an error condition, the processing is halted and the output index is set to the array index of the offending Coordinate3D instance. When successful, the output index is set to the size of the array plus one.
Inputs	source_srf: BaseSRF3D source_coordinate_array: Coordinate3D_Array h_st: ORM_Transformation_Parameters
Outputs	target_coordinate_array: Coordinate3D_Array index: Integer_Unsigned

Element	Specification
Class	BaseSRF3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_INPUT if source_coordinate_array is not a valid Coordinate3D_Array data structure. 3) INVALID_PARAMETERS if the input h_st parameter values are not valid, or are not of the correct dimension. 4) INVALID_SOURCE_COORDINATE if the Coordinate3D instance at index in source_coordinate_array is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 5) INVALID_TARGET_COORDINATE if the spatial position of the Coordinate3D instance at index is not in the accuracy domain of this SRF.
Abstract method	ChangeCoordinate3DSRF
Semantics	Changes the SRF representation of the spatial position specified by the input Coordinate3D source_coordinate in the source SRF source_srf to a Coordinate3D target_coordinate in this SRF, the target SRF, in accordance with 10.4.2 using the implicit ORM transformation $\boldsymbol{H}_{\mathbf{ST}}$ given in Equation (10.2). Equation (10.2) assumes both SRFs are based on object-fixed ORMs for the same spatial object.
Inputs	source_srf: BaseSRF3D source_coordinate: Coordinate3D
Outputs	target_coordinate: Coordinate3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_coordinate is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 4) INVALID_TARGET_COORDINATE if the spatial position of source_coordinate is not in the accuracy domain of this SRF.
Abstract method	ChangeCoordinate3DSRFObject
Semantics	Changes the SRF representation of the spatial position specified by the input Coordinate3D source_coordinate in the source SRF source_srf to a Coordinate3D target_coordinate in this SRF, the target SRF, using an explicit ORM transformation $\boldsymbol{H}_{\mathbf{ST}}$ as specified by input h_st in accordance with 10.4.2. The input h_st is required in the case of SRFs for different spatial objects or in the case that one or both ORM reference transformations have not been specified.
Inputs	source_srf: BaseSRF3D source_coordinate: Coordinate3D h_st: ORM_Transformation_Parameters
Outputs	target_coordinate: Coordinate3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_coordinate is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_PARAMETERS if the input h_st parameter values are not valid, or are not of the correct dimension. 4) INVALID_TARGET_COORDINATE if the spatial position of source_coordinate is not in the accuracy domain of this SRF.

Element	Specification
Class	BaseSRF3D
Abstract method	ChangeDirectionArraySRF
Semantics	Performs the same operation defined for the ChangeDirectionSRF method on each Direction instance in source_direction_array. The processing is in array indexing order. Upon an error condition, the processing is halted and the output index is set to the array index of the offending Direction instance. When successful, the output index is set to the size of the array plus one.
Inputs	source_srf: BaseSRF3D source_direction_array: Direction_Array
Outputs	target_direction_array: Direction_Array index: Integer_Unsigned
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_INPUT if source_direction_array is not a valid Direction_Array data structure. 3) INVALID_SOURCE_DIRECTION if (1) the Direction instance at index in source_direction_array is not a valid Direction instance, (2) the reference coordinate of the Direction instance at index in source_direction_array is not associated with source_srf, or (3) the reference coordinate of the Direction instance at index in source_direction_array is not in the accuracy domain of source_srf. 4) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 5) INVALID_TARGET_DIRECTION if the spatial position of the reference coordinate of the Direction instance at index is not in the accuracy domain of this SRF.
Abstract method	ChangeDirectionArraySRFObject
Semantics	Performs the same operation defined for the ChangeDirectionSRFObject method on each Direction instance in source_direction_array. The processing is in array indexing order. Upon an error condition, the processing is halted and the output index is set to the array index of the offending Direction instance. When successful, the output index is set to the size of the array plus one.
Inputs	source_srf: BaseSRF3D source_direction_array: Direction_Array h_st: ORM_Transformation_Parameters
Outputs	target_direction_array: Direction_Array index: Integer_Unsigned

Element	Specification
Class	BaseSRF3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_INPUT if source_direction_array is not a valid Direction_Array data structure. 3) INVALID_PARAMETERS if the input h_st parameter values are not valid, or are not of the correct dimension. 4) INVALID_SOURCE_DIRECTION if (1) the Direction instance at index in source_direction_array is not a valid Direction instance, (2) the reference coordinate of the Direction instance at index in source_direction_array is not associated with source_srf, or (3) the reference coordinate of the Direction instance at index in source_direction_array is not in the accuracy domain of source_srf. 5) INVALID_TARGET_DIRECTION if the spatial position of the reference coordinate of the Direction instance at index is not in the accuracy domain of this SRF.
Abstract method	ChangeDirectionSRF
Semantics	Changes the SRF representation of the input source_direction, a spatial direction represented in the source SRF source_srf, to its corresponding representation, target_direction, in this SRF, the target SRF. The output target_direction is computed in accordance with 10.5.5 using the implicit ORM transformation $\boldsymbol{H}_{\mathbf{ST}}$ given in Equation (10.2). The reference coordinate of the output target_direction is computed from the reference coordinate of the input source_direction using the functionality of ChangeCoordinate3DSRF.
Inputs	source_srf: BaseSRF3D source_direction: Direction
Outputs	target_direction: Direction
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_DIRECTION if (1) source_direction is not a valid Direction instance, (2) the reference coordinate of source_direction is not associated with source_srf, or (3) the reference coordinate of source_direction is not in the accuracy domain of source_srf. 3) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 4) INVALID_TARGET_DIRECTION if the spatial position of the reference coordinate of source_direction is not in the accuracy domain of this SRF.
Abstract method	ChangeDirectionSRFObject
Semantics	Changes the SRF representation of the input source_direction, a spatial direction represented in the source SRF source_srf, to its corresponding representation, target_direction, in this SRF, the target SRF. The output target_direction is computed in accordance with 10.5.5 using the rotation matrix of the explicit ORM transformation $\boldsymbol{H}_{\mathbf{ST}}$ as specified by input h_st. The reference coordinate of the output target_direction is computed from the reference coordinate of the input source_direction using the functionality of ChangeCoordinate3DSRF.

Element	Specification
Class	BaseSRF3D
Inputs	source_srf: BaseSRF3D source_direction: Direction h_st: ORM_Transformation_Parameters
Outputs	target_direction: Direction
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_DIRECTION if (1) source_direction is not a valid Direction instance, (2) the reference coordinate of source_direction is not associated with source_srf, or (3) the reference coordinate of source_direction is not in the accuracy domain of source_srf. 3) INVALID_PARAMETERS if the input h_st parameter values are not valid, or are not of the correct dimension. 4) INVALID_TARGET_DIRECTION if the spatial position of the reference coordinate of source_direction is not in the accuracy domain of this SRF.
Abstract method	ComputeSRFOrientation
Semantics	Creates an Orientation instance representing the orientation of the local tangent frame SRF at source_ref_location in source_srf with respect to the local tangent frame SRF at target_ref_location in this SRF (see 10.5.5).
Inputs	source_srf: BaseSRF3D source_ref_location: Coordinate3D target_ref_location: Coordinate3D
Outputs	out_orientation: Orientation
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_ref_location is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_TARGET_COORDINATE if target_ref_location is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 4) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED).
Abstract method	CreateCoordinate3D
Semantics	Creates a Coordinate3D instance associated with this SRF from three ordered coordinate-component values. Coordinate-components that represent lengths shall be evaluated as metres. Coordinate-components that represent angles shall be evaluated as radians.
Inputs	first_coordinate_component: Long_Float second_coordinate_component: Long_Float third_coordinate_component: Long_Float
Outputs	new_coordinate: Coordinate3D
Error conditions	INVALID_INPUT if the spatial position specified by the input coordinate-component values is not in the accuracy domain of this SRF.
Abstract method	CreateDirection
Semantics	Creates a Direction instance in this SRF with the specified reference coordinate and vector components. The input vector shall be a unit vector.

Element	Specification
Class	BaseSRF3D
Inputs	reference_coordinate: Coordinate3D first_direction_component: Long_Float second_direction_component: Long_Float third_direction_component: Long_Float
Output	new_direction: Direction
Error conditions	1) INVALID_COORDINATE if reference_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INCOMPATIBLE_INPUTS if the direction components do not specify a unit vector.
Abstract method	CreateLococentricEuclidean3DSRF
Semantics	Creates a LococentricEuclidean3D SRF with origin at lococentre and orthogonal axes determined by the input primary_axis and secondary_axis directions. The created SRF has the same ORM and RT code as this SRF.
Inputs	lococentre: Coordinate3D primary_axis: Direction secondary_axis: Direction
Outputs	lococentricEuclidean3D_srf: LococentricEuclidean3D
Error conditions	1) INVALID_COORDINATE if lococentre is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_PRIMARY_AXIS_DIRECTION if (1) primary_axis is not a valid Direction instance, (2) the reference coordinate of primary_axis is not associated with this SRF, or (3) the reference coordinate of primary_axis is not in the accuracy domain of this SRF. 3) INVALID_SECONDARY_AXIS_DIRECTION if (1) secondary_axis is not a valid Direction instance, (2) the reference coordinate of secondary_axis is not associated with this SRF, or (3) the reference coordinate of secondary_axis is not in the accuracy domain of this SRF. 4) INCOMPATIBLE_INPUTS if primary_axis and secondary_axis are not orthogonal directions.
Abstract method	EuclideanDistance
Semantics	Outputs the Euclidean distance in metres between the spatial points represented by Coordinate3D instances point1_coordinate and point2_coordinate (see 10.6).
Inputs	point1_coordinate: Coordinate3D point2_coordinate: Coordinate3D
Outputs	distance: Long_Float
Error conditions	1) INVALID_POINT1_COORDINATE if point1_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_POINT2_COORDINATE if point2_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	GetCoordinate3DValues
Semantics	Retrieves the three ordered coordinate-components of a Coordinate3D instance. Coordinate-components that represent lengths shall be expressed in metres. Coordinate-components that represent angles shall be expressed in radians.
Inputs	coordinate: Coordinate3D

Element	Specification
Class	BaseSRF3D
Outputs	first_coordinate_component: Long_Float second_coordinate_component: Long_Float third_coordinate_component: Long_Float
Error conditions	INVALID_COORDINATE if coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	GetDirectionValues
Semantics	Retrieves the reference coordinate and vector components from a Direction instance.
Inputs	direction: Direction
Outputs	reference_coordinate: Coordinate3D first_direction_component: Long_Float second_direction_component: Long_Float third_direction_component: Long_Float
Error conditions	INVALID_DIRECTION if (1) direction is not a valid Direction instance, (2) the reference coordinate of direction is not associated with this SRF, or (3) the reference coordinate of direction is not in the accuracy domain of this SRF.
Abstract method	GetLocalTangentFrameSRFParameters
Semantics	Computes the parameters corresponding to the local tangent frame SRF at reference_location.
Inputs	reference_location: Coordinate3D
Outputs	ltf_parameters: LCE_3D_Parameters
Error conditions	INVALID_SOURCE_COORDINATE if reference_location is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	GetSRFRegion
Semantics	Returns the SRF region and extended SRF region for this SRF.
Inputs	none
Outputs	region_type: SRF_Region_Type first_component_interval: Interval second_component_interval: Interval third_component_interval: Interval extended_first_component_interval: Interval extended_second_component_interval: Interval extended_third_component_interval: Interval
Error conditions	No additional error conditions.
Abstract method	InRTRegionTest
Semantics	Determines whether the specified coordinate is within the RT region for this SRF. If the RT region is specified in this International Standard, is_set is returned as true, and in_region is returned as true if coordinate is within the RT region. If the RT region is not specified, is_set is returned as false, and in_region is returned as true.
Inputs	coordinate: Coordinate3D
Outputs	is_set: Boolean in_region: Boolean

Element	Specification
Class	BaseSRF3D
Error conditions	INVALID_COORDINATE if coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	InSRFRegionTest
Semantics	Determines whether the specified coordinate is within the SRF region and/or extended SRF region for this SRF.
Inputs	coordinate: Coordinate3D
Outputs	status: SRF_Region_Status
Error conditions	INVALID_COORDINATE if coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	InSRFRegionTestArray
Semantics	Performs the same operation defined for the InSRFRegionTest method on each Coordinate3D instance in coordinate_array. The processing is in array indexing order. Upon an error condition, the processing is halted and the output index is set to the array index of the offending Coordinate3D instance. When successful, the output index is set to the size of the array plus one.
Inputs	coordinate_array: Coordinate3D_Array
Outputs	status_array: SRF_Region_Status_Array index: Integer_Unsigned
Error conditions	1) INVALID_INPUT if coordinate_array is not a valid Coordinate3D array data structure. 2) INVALID_COORDINATE if the Coordinate3D instance at index in coordinate_array is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	SetSRFRegion
Semantics	Sets the SRF region and extended SRF region for this SRF. (See 8.3.2.4.)
Inputs	region_type: SRF_Region_Type first_component_interval: Interval second_component_interval: Interval third_component_interval: Interval extended_first_component_interval: Interval extended_second_component_interval: Interval extended_third_component_interval: Interval
Outputs	none

Element	Specification
Class	BaseSRF3D
Error conditions	1) INVALID_INPUT if a) a region_type input value of GEODETIC_REGION is specified, but this SRF has no corresponding Celestiodetic SRF, or b) an interval coordinate-component is not angular, and the value of lower_bound is not strictly less than the value of upper_bound, and interval_type is not a semi-interval or unbounded type, or c) an interval coordinate-component is angular, and the value of interval_type is a semi-interval type, or d) an interval coordinate-component is angular, and the value of interval_type is a bounded interval, and the value of lower_bound or upper_bound does not satisfy the corresponding CS domain constraints, or the values are equal. 2) INCOMPATIBLE_INPUTS if any of the extended SRF region coordinate-component intervals does not contain the corresponding SRF region coordinate-component interval.
Abstract method	TransformOrientation
Semantics	Given source_orientation, an orientation of a spatial object with respect to the local tangent frame SRF (LTF_S) at source_ref_location in source_srf, this method computes the orientation of the spatial object with respect to the local tangent frame SRF (LTF_T) at target_ref_location in this SRF.
Inputs	source_srf: BaseSRF3D source_ref_location: Coordinate3D source_orientation: Orientation target_ref_location: Coordinate3D
Outputs	target_orientation: Orientation
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_ref_location is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_ORIENTATION if source_orientation is not a valid Orientation instance. 4) INVALID_TARGET_COORDINATE if target_ref_location is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 5) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED).
Abstract method	TransformOrientationCommonOrigin
Semantics	Given source_orientation, an orientation of a spatial object with respect to the local tangent frame SRF (LTF_S) at source_ref_location in source_srf, this method computes the orientation of the spatial object with respect to the local tangent frame SRF (LTF_T) at target_ref_location in this SRF. The output target_ref_location represents the same spatial position as the input source_ref_location.
Inputs	source_srf: BaseSRF3D source_ref_location: Coordinate3D source_orientation: Orientation
Outputs	target_ref_location: Coordinate3D target_orientation: Orientation

Element	Specification
Class	BaseSRF3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_ref_location is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_ORIENTATION if source_orientation is not a valid Orientation instance. 4) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 5) INVALID_TARGET_COORDINATE if the spatial position of source_ref_location is not in the accuracy domain of this SRF.
Abstract method	TransformVector
Semantics	Given source_vector, a vector in the local tangent frame SRF (LTF_S) at source_ref_location in source_srf, this method computes the vector in the local tangent frame SRF (LTF_T) at target_ref_location in this SRF (see 10.5.6).
Inputs	source_srf: BaseSRF3D source_ref_location: Coordinate3D source_vector: Vector_3D target_ref_location: Coordinate3D
Outputs	target_vector: Vector_3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_ref_location is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_VECTOR if source_vector is not a valid Vector_3D data structure. 4) INVALID_TARGET_COORDINATE if target_ref_location is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 5) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED).
Abstract method	TransformVectorCommonOrigin
Semantics	Given source_vector, a vector in the local tangent frame SRF (LTF_S) at source_ref_location in source_srf, this method computes the vector in the local tangent frame SRF (LTF_T) at target_ref_location in this SRF. The output target_ref_location represents the same spatial position as the input source_ref_location.
Inputs	source_srf: BaseSRF3D source_ref_location: Coordinate3D source_vector: Vector_3D
Outputs	target_ref_location: Coordinate3D target_vector: Vector_3D

Element	Specification
Class	BaseSRF3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_ref_location is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_VECTOR if source_vector is not a valid Vector_3D data structure. 4) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 5) INVALID_TARGET_COORDINATE if the spatial position of source_ref_location is not in the accuracy domain of this SRF.
Abstract method	TransformVectorInBodyFrame
Semantics	Given body_vector, a vector in a body frame, and body_orientation, the orientation of the body frame with respect to the local tangent frame SRF (LTF_S) at source_ref_location in source_srf, this method computes the vector in the local tangent frame SRF (LTF_T) at target_ref_location in this SRF.
Inputs	source_srf: BaseSRF3D source_ref_location: Coordinate3D body_orientation: Orientation body_vector: Vector_3D target_ref_location: Coordinate3D
Outputs	target_vector: Vector_3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_ref_location is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_ORIENTATION if body_orientation is not a valid Orientation instance. 4) INVALID_VECTOR if body_vector is not a valid Vector_3D data structure. 5) INVALID_TARGET_COORDINATE if target_ref_location is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 6) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED).
Abstract method	TransformVectorInBodyFrameCommonOrigin
Semantics	Given body_vector, a vector in a body frame, and body_orientation, the orientation of the body frame with respect to the local tangent frame SRF (LTF_S) at source_ref_location in source_srf, this method computes the vector in the local tangent frame SRF (LTF_T) at target_ref_location in this SRF. The output target_ref_location represents the same spatial position as the input source_ref_location.
Inputs	source_srf: BaseSRF3D source_ref_location: Coordinate3D body_orientation: Orientation body_vector: Vector_3D
Outputs	target_ref_location: Coordinate3D target_vector: Vector_3D

Element	Specification
Class	BaseSRF3D
Error conditions	1) INVALID_SOURCE_SRF if source_srf is not a valid SRF. 2) INVALID_SOURCE_COORDINATE if source_ref_location is (1) not associated with source_srf, or (2) not in the accuracy domain of source_srf. 3) INVALID_ORIENTATION if body_orientation is not a valid Orientation instance. 4) INVALID_VECTOR if body_vector is not a valid Vector_3D data structure. 5) OPERATION_UNSUPPORTED if (1) source_srf and this SRF are for different spatial objects, or (2) the ORMs of source_srf and this SRF are different, and either source_srf or this SRF was created with reference transformation RT_Code value 0 (UNSPECIFIED). 6) INVALID_TARGET_COORDINATE if the spatial position of source_ref_location is not in the accuracy domain of this SRF.

11.3.5.4 BaseSRFMapProjection

BaseSRFMapProjection is a subclass of BaseSRFwithEllipsoidalHeight. This abstract class has six concrete subclasses, as shown in Figure 11.5. This abstract class adds the following methods which are specified in Table 11.14:

ConvergenceOfTheMeridian,
MapAzimuth, and
PointDistortion.

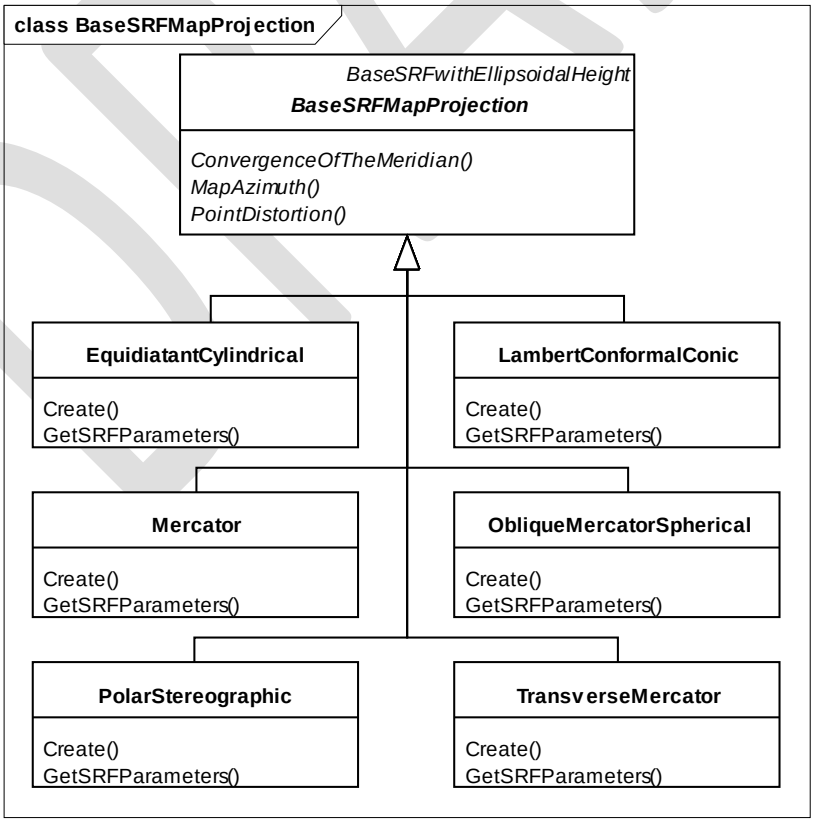

Figure 11.5 — BaseSRFMapProjection class hierarchy

Table 11.14 — BaseSRFMapProjection

Element	Specification
Class	BaseSRFMapProjection
Description	An abstract subclass of BaseSRFwithEllipsoidalHeight specifying the common elements of map projection SRFs.
Superclass(es)	LifeCycleObject: Create, Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithEllipsoidalHeight: CreateLocalTangentSpaceEuclideanSRF, CreateSurfaceCoordinate, EuclideanDistance, GeodesicDestination, GeodesicDistance, GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues, InRTRegionTest, PromoteSurfaceCoordinate, TruncateCoordinate3D, VerticalOffset
Abstract method	ConvergenceOfTheMeridian
Semantics	Outputs the convergence of the meridian (COM) in radians at a position on the surface of the ellipsoid RD (see 5.8.3.5).
Inputs	surface_coordinate: SurfaceCoordinate
Outputs	gamma: Long_Float
Error conditions	INVALID_COORDINATE if surface_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	MapAzimuth
Semantics	Outputs the map azimuth in radians at the point1_coordinate towards the point2_coordinate (see 5.8.3.4).
Inputs	point1_coordinate: SurfaceCoordinate point2_coordinate: SurfaceCoordinate
Outputs	azimuth: Long_Float
Error conditions	1) INVALID_POINT1_COORDINATE if point1_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_POINT2_COORDINATE if point2_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	PointDistortion
Semantics	Outputs the point distortion at a position on the surface of the ellipsoid RD (see 5.8.3.3).
Inputs	surface_coordinate: SurfaceCoordinate

Element	Specification
Class	`BaseSRFMapProjection`
Outputs	`distortion: Long_Float`
Error conditions	`INVALID_COORDINATE` if `surface_coordinate` is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.

11.3.5.5 BaseSRFwithEllipsoidalHeight

`BaseSRFwithEllipsoidalHeight` is a subclass of `BaseSRF3D`. This abstract class represents SRFs for which the zero-value vertical coordinate-component is the surface of the oblate ellipsoid RD of the ORM of the SRF (see 8.4). This abstract class has one abstract subclass, and two concrete subclasses, as shown in Figure 11.6. This abstract class adds the following methods which are specified in Table 11.15:

`CreateLocalTangentSpaceEuclideanSRF`,
`CreateSurfaceCoordinate`,
`EuclideanDistance`,
`GeodesicDestination`,
`GeodesicDistance`,
`GeodesicDistanceWithAzimuths`,
`GetSurfaceCoordinateValues`,
`InRTRegionTest`,
`PromoteSurfaceCoordinate`,
`TruncateCoordinate3D`, and
`VerticalOffset`.

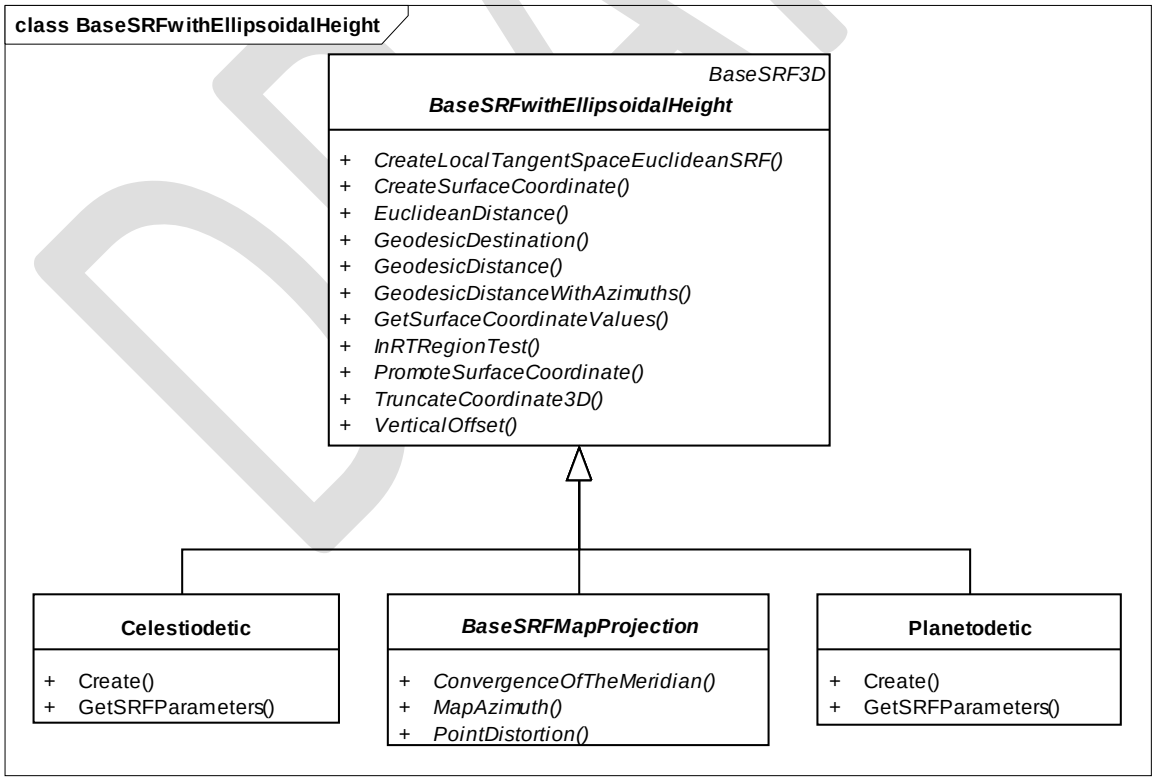

Figure 11.6 — BaseSRFwithEllipsoidalHeight class hierarchy

Table 11.15 — BaseSRFwithEllipsoidalHeight

Element	Specification
Class	BaseSRFwithEllipsoidalHeight
Description	An abstract class representing the common elements of BaseSRF3D concrete subclasses for which the zero-value vertical coordinate-component is the surface of the oblate ellipsoid RD of the ORM of the SRF.
Superclass(es)	LifeCycleObject: Create, Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Abstract method	CreateLocalTangentSpaceEuclideanSRF
Semantics	Creates a LocalTangentSpaceEuclidean SRF at the location given by surface_coordinate, with its Y axis aligned with azimuth, and with its origin offset by false_x_origin, false_y_origin and offset_height. The created SRF has the same ORM as this SRF. Input parameters that represent lengths shall be evaluated as metres. The azimuth parameter shall be evaluated as radians.
Inputs	surface_coordinate: SurfaceCoordinate azimuth: Long_Float false_x_origin: Long_Float false_y_origin: Long_Float offset_height: Long_Float
Outputs	localtangentEuclidean_srf: LocalTangentSpaceEuclidean
Error conditions	INVALID_COORDINATE if surface_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	CreateSurfaceCoordinate
Semantics	Creates a surface coordinate on the ellipsoid RD surface. Coordinate-components that represent lengths shall be evaluated as metres. Coordinate-components that represent angles shall be evaluated as radians.
Inputs	first_coordinate_component: Long_Float second_coordinate_component: Long_Float
Outputs	new_coordinate: SurfaceCoordinate
Error conditions	INVALID_INPUT if the spatial position specified by the input coordinate-component values is not in the accuracy domain of this SRF.
Abstract method	EuclideanDistance

Element	Specification
Class	BaseSRFwithEllipsoidalHeight
Semantics	Outputs the Euclidean distance in metres between the spatial points represented by the SurfaceCoordinate instances point1_coordinate and point2_coordinate (see 10.6).
Inputs	point1_coordinate: SurfaceCoordinate point2_coordinate: SurfaceCoordinate
Outputs	distance: Long_Float
Error conditions	1) INVALID_POINT1_COORDINATE if point1_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_POINT2_COORDINATE if point2_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	GeodesicDestination
Semantics	Outputs the destination position on the surface of the ellipsoid RD, and also provides the forward azimuth in radians at the destination position, given the starting position, the forward azimuth in radians at the starting position, and the distance in metres to the destination (*i.e.*, solves the geodesic direct problem, see 10.7.3).
Inputs	point1_coordinate: SurfaceCoordinate point1_forward_azimuth: Long_Float distance: Long_Float
Outputs	point2_coordinate: SurfaceCoordinate point2_forward_azimuth: Long_Float
Error conditions	1) INVALID_POINT1_COORDINATE if point1_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_INPUT if the distance value is non-positive, or exceeds 95% of a meridian length on the applicable oblate ellipsoid RD.
Abstract method	GeodesicDistance
Semantics	Outputs the geodesic distance in metres between a pair of positions on the surface of the ellipsoid RD (*i.e.*, solves the geodesic indirect problem, see 10.7.4).
Inputs	point1_coordinate: SurfaceCoordinate point2_coordinate: SurfaceCoordinate
Outputs	distance: Long_Float
Error conditions	1) INVALID_POINT1_COORDINATE if point1_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_POINT2_COORDINATE if point2_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	GeodesicDistanceWithAzimuths
Semantics	Outputs the geodesic distance in metres between a pair of positions on the surface of the ellipsoid RD, and also provides the forward azimuths in radians at both positions (*i.e.*, solves the geodesic indirect problem, see 10.7.4).
Inputs	point1_coordinate: SurfaceCoordinate point2_coordinate: SurfaceCoordinate
Outputs	distance: Long_Float point1_forward_azimuth: Long_Float point2_forward_azimuth: Long_Float

Element	Specification
Class	BaseSRFwithEllipsoidalHeight
Error conditions	1) INVALID_POINT1_COORDINATE if point1_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_POINT2_COORDINATE if point2_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	GetSurfaceCoordinateValues
Semantics	Retrieves the coordinate-component values of a surface coordinate on the ORM surface. Coordinate-components that represent lengths shall be expressed in metres. Coordinate-components that represent angles shall be expressed in radians.
Inputs	surface_coordinate: SurfaceCoordinate
Outputs	first_coordinate_component: Long_Float second_coordinate_component: Long_Float
Error conditions	INVALID_COORDINATE if surface_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	InRTRegionTest
Semantics	Determines whether the specified coordinate is within the RT region for this SRF. If the RT region is specified in this International Standard, is_set is returned as true, and in_region is returned as true if coordinate is within the RT region. If the RT region is not specified, is_set is returned as false, and in_region is returned as true.
Inputs	coordinate: Surface_Coordinate
Outputs	is_set: Boolean in_region: Boolean
Error conditions	INVALID_COORDINATE if coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	PromoteSurfaceCoordinate
Semantics	Creates a Coordinate3D instance representing the same location as specified by surface_coordinate (promote surface coordinate to coordinate 3D, see 10.4.3).
Inputs	surface_coordinate: SurfaceCoordinate
Outputs	coordinate: Coordinate3D
Error conditions	INVALID_COORDINATE if surface_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	TruncateCoordinate3D
Semantics	Creates the SurfaceCoordinate instance associated with coordinate (truncate to surface; see 10.4.3).
Inputs	coordinate: Coordinate3D
Outputs	surface_coordinate: SurfaceCoordinate
Error conditions	INVALID_COORDINATE if coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	VerticalOffset
Semantics	Outputs the vertical offset (see 9.3), at the input surface_coordinate, between the surface of the ellipsoid RD of this SRF and the DSS specified by the input dss_code. If the value of dss_code is 0 (UNSPECIFIED), the output offset value shall be 0.

Element	Specification
Class	`BaseSRFwithEllipsoidalHeight`
Inputs	`dss_code:` `DSS_Code` `surface_coordinate:` `SurfaceCoordinate`
Outputs	`offset:` `Long_Float`
Error conditions	1) `UNDEFINED_CODE` if `dss_code` is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) `INVALID_COORDINATE` if `surface_coordinate` is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 3) `OPERATION_UNSUPPORTED` if the DSS is not a VOS for this SRF, if the DSS does not have a supported DSS model, or if the vertical offset is undefined at `surface_coordinate`.

11.3.5.6 BaseSRFwithTangentPlaneSurface

`BaseSRFwithTangentPlaneSurface` is a subclass of `BaseSRF3D`. This abstract class represents SRFs for which the zero-value vertical coordinate-component surface is parallel to the plane tangent to the oblate ellipsoid RD of the ORM of the SRF at a specified point (see 8.4). This abstract class has three concrete subclasses, as shown in Figure 11.7. This abstract class adds the following methods which are specified in Table 11.16:

`CreateSurfaceCoordinate`,
`EuclideanDistance`,
`GetSurfaceCoordinateValues`,
`PromoteSurfaceCoordinate`, and
`TruncateCoordinate3D`.

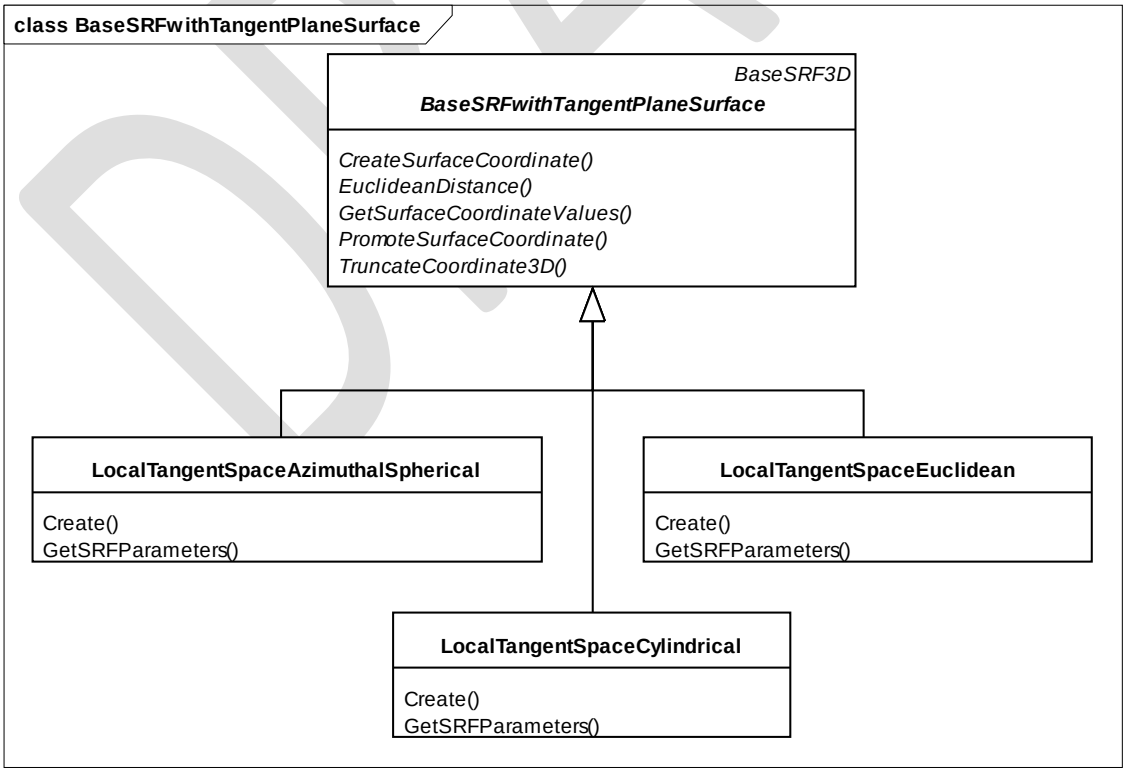

Figure 11.7 — BaseSRFwithTangentPlaneSurface class hierarchy

Table 11.16 — BaseSRFwithTangentPlaneSurface

Element	Specification
Class	BaseSRFwithTangentPlaneSurface
Description	An abstract class representing the common elements of BaseSRF3D concrete subclasses for which the zero-value vertical coordinate-component surface is parallel to the plane tangent to the oblate ellipsoid RD of the ORM of the SRF at a specified point.
Superclass(es)	LifeCycleObject: Create, Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Abstract method	CreateSurfaceCoordinate
Semantics	Creates a surface coordinate on the tangent plane surface. Coordinate-components that represent lengths shall be evaluated as metres. Coordinate-components that represent angles shall be evaluated as radians.
Inputs	first_coordinate_component: Long_Float second_coordinate_component: Long_Float
Outputs	new_coordinate: SurfaceCoordinate
Error conditions	INVALID_INPUT if the spatial position specified by the input coordinate-component values is not in the accuracy domain of this SRF.
Abstract method	EuclideanDistance
Semantics	Outputs the Euclidean distance in metres between the spatial points represented by SurfaceCoordinate instances point1_coordinate and point2_coordinate (see 10.6).
Inputs	point1_coordinate: SurfaceCoordinate point2_coordinate: SurfaceCoordinate
Outputs	distance: Long_Float
Error conditions	1) INVALID_POINT1_COORDINATE if point1_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF. 2) INVALID_POINT2_COORDINATE if point2_coordinate is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	GetSurfaceCoordinateValues
Semantics	Retrieves the coordinate-component values of a surface coordinate on the tangent plane surface. Coordinate-components that represent lengths shall be expressed in metres. Coordinate-components that represent angles shall be expressed in radians.
Inputs	surface_coordinate: SurfaceCoordinate

Element	Specification
Class	`BaseSRFwithTangentPlaneSurface`
Outputs	`first_coordinate_component: Long_Float` `second_coordinate_component: Long_Float`
Error conditions	`INVALID_COORDINATE` if `surface_coordinate` is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	`PromoteSurfaceCoordinate`
Semantics	Creates a `Coordinate3D` instance representing the same location as specified by `surface_coordinate` (promote surface coordinate to 3D coordinate, see 10.4.3).
Inputs	`surface_coordinate: SurfaceCoordinate`
Outputs	`coordinate: Coordinate3D`
Error conditions	`INVALID_COORDINATE` if `surface_coordinate` is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.
Abstract method	`TruncateCoordinate3D`
Semantics	Creates the `SurfaceCoordinate` instance associated with `coordinate` (truncate to surface; see 10.4.3).
Inputs	`coordinate: Coordinate3D`
Outputs	`surface_coordinate: SurfaceCoordinate`
Error conditions	`INVALID_COORDINATE` if `coordinate` is (1) not associated with this SRF, or (2) not in the accuracy domain of this SRF.

11.3.5.7 Orientation

The class `Orientation` is an abstract class that represents the orientation of a spatial object with respect to a reference. This abstract class has six concrete subclasses, as shown in Figure 11.8. It provides the following methods that are common to all orientation representations:

```
ComposeWith,
GetAxisAngle,
GetEulerAnglesZXZ,
GetMatrix3x3,
GetQuaternion,
GetTaitBryanAnglesXYZ,
GetTaitBryanAnglesZYX,
TransformVector.
```

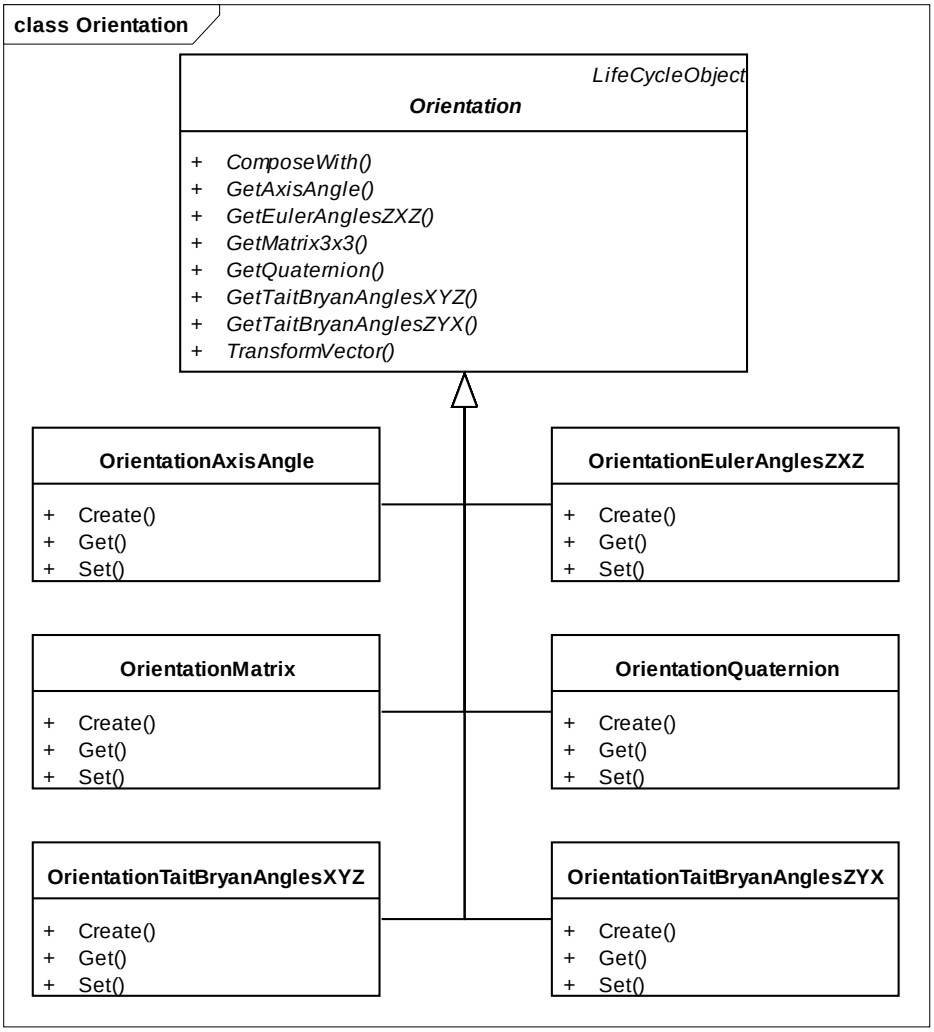

Figure 11.8 — Orientation class hierarchy

Table 11.17 — Orientation

Element	Specification
Class	`Orientation`
Description	An abstract class that represents the orientation of a spatial object with respect to a reference. It provides methods that are common to all orientation representations.
Superclass(es)	`LifeCycleObject`: `Create`, `Destroy`
Abstract method	`ComposeWith`
Semantics	This method composes two `Orientation` instances and returns the resulting `Orientation` instance. Thus, if S_1, S_2, and S_3 are three spatial objects, and the orientation of S_1 with respect to S_2 is the input `orientation_1_2`, and the orientation of S_2 with respect to S_3 is this `Orientation` instance, then the orientation of S_1 with respect to S_3 is returned as `orientation_1_3`. (See 10.5.7.)
Inputs	`orientation_1_2: Orientation`

Element	Specification
Class	`Orientation`
Outputs	`orientation_1_3:  Orientation`
Error conditions	No additional error conditions.
Abstract method	`GetAxisAngle`
Semantics	This method returns the representation of the `Orientation` instance in the form of a unit vector ***n*** (with components n_1, n_2, and n_3) and a rotation angle. (See 6.7.3.)
Inputs	none
Outputs	`axis_angle:  Axis_Angle_Parameters`
Error conditions	No additional error conditions.
Abstract method	`GetEulerAnglesZXZ`
Semantics	This method returns the representation of the `Orientation` instance in the form of three consecutive Euler rotation angles about the principal coordinate system axes (see 6.7.4.3): $$\boldsymbol{R}_{z''}(\text{precession}) \circ \boldsymbol{R}_{x'}(\text{nutation}) \circ \boldsymbol{R}_{z}(\text{spin}) = \boldsymbol{R}_{z}(\text{spin}) \circ \boldsymbol{R}_{x}(\text{nutation}) \circ \boldsymbol{R}_{z}(\text{precession}).$$
Inputs	none
Outputs	`euler_angles_ZXZ:  Euler_Angles_ZXZ_Parameters`
Error conditions	No additional error conditions.
Abstract method	`GetMatrix3x3`
Semantics	This method returns the representation of the `Orientation` instance in the form of a 3x3 rotation matrix. (See 6.7.2.)
Inputs	none
Outputs	`matrix:  Matrix_3x3`
Error conditions	No additional error conditions.
Abstract method	`GetQuaternion`
Semantics	This method returns the representation of the `Orientation` instance in the form of a quaternion. (See 6.7.5.)
Inputs	none
Outputs	`quaternion:  Quaternion_Parameters`
Error conditions	No additional error conditions.
Abstract method	`GetTaitBryanAnglesXYZ`
Semantics	This method returns the representation of the `Orientation` instance in the form of three principal rotation angles specifying body-fixed principal rotation composition (see 6.7.4.4): $$\boldsymbol{R}_{z''}(\text{yaw}) \circ \boldsymbol{R}_{y'}(\text{pitch}) \circ \boldsymbol{R}_{x}(\text{roll}) = \boldsymbol{R}_{x}(\text{roll}) \circ \boldsymbol{R}_{y}(\text{pitch}) \circ \boldsymbol{R}_{z}(\text{yaw}).$$
Inputs	none
Outputs	`tait_bryanXYZ:  Tait_Bryan_Parameters`
Error conditions	No additional error conditions.

Element	Specification
Class	`Orientation`
Abstract method	`GetTaitBryanAnglesZYX`
Semantics	This method returns the representation of the `Orientation` instance in the form of three principal rotation angles specifying body-fixed principal rotation composition (see 6.7.4.4): $$\boldsymbol{R}_{x''}(\text{roll}) \circ \boldsymbol{R}_{y'}(\text{pitch}) \circ \boldsymbol{R}_{z}(\text{yaw}) = \boldsymbol{R}_{z}(\text{yaw}) \circ \boldsymbol{R}_{y}(\text{pitch}) \circ \boldsymbol{R}_{x}(\text{roll})\,.$$
Inputs	none
Outputs	`tait_bryanZYX:` `Tait_Bryan_Parameters`
Error conditions	No additional error conditions.
Abstract method	`TransformVector`
Semantics	This method transforms the input vector, represented in the source reference frame, to its representation in the target reference frame, where this `Orientation` instance specifies the orientation of the source reference frame with respect to the target reference frame. (See 10.5.6.)
Inputs	`input_vector:` `Vector_3D`
Outputs	`output_vector:` `Vector_3D`
Error conditions	No additional error conditions.

11.3.6 SRF concrete subclasses of BaseSRF2D

11.3.6.1 Introduction

The concrete subclasses of `BaseSRF2D` are:

`LocalSpaceAzimuthal`,
`LocalSpacePolar`, and
`LocalSpaceRectangular2D`.

These concrete classes override the `Create` method of `LifeCycleObject` so that an instance of the class can be created. In those cases for which the `Create` method requires additional SRF-specific parameters, a `GetSRFParameters` method is also specified for the class.

11.3.6.2 LocalSpaceAzimuthal

Table 11.18 — LocalSpaceAzimuthal

Element	Specification
Class	`LocalSpaceAzimuthal`
Description	An instance of this class corresponds to an instance of SRFT LOCAL_SPACE_AZIMUTHAL_2D

Element	Specification
Class	LocalSpaceAzimuthal
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF2D: ChangeCoordinate2DArraySRF, ChangeCoordinate2DArraySRFObject, ChangeCoordinate2DSRF, ChangeCoordinate2DSRFObject, CreateCoordinate2D, EuclideanDistance, GetCoordinate2DValues, InRTRegionTest
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a LocalSpaceAzimuthal SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: LocalSpaceAzimuthal
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LOCAL_SPACE_AZIMUTHAL_2D, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.6.3 LocalSpacePolar

Table 11.19 — LocalSpacePolar

Element	Specification
Class	LocalSpacePolar
Description	An instance of this class corresponds to an instance of SRFT LOCAL_SPACE_POLAR_2D.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF2D: ChangeCoordinate2DArraySRF, ChangeCoordinate2DArraySRFObject, ChangeCoordinate2DSRF, ChangeCoordinate2DSRFObject, CreateCoordinate2D, EuclideanDistance, GetCoordinate2DValues, InRTRegionTest
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a LocalSpacePolar SRF corresponding to the input ORM_Code parameter. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code

Outputs	new_srf: LocalSpacePolar
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LOCAL_SPACE_POLAR_2D, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.6.4 LocalSpaceRectangular2D

Table 11.20 — LocalSpaceRectangular2D

Element	Specification
Class	LocalSpaceRectangular2D
Description	An instance of this class corresponds to an instance of SRFT LOCAL_SPACE_RECTANGULAR_2D.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF2D: ChangeCoordinate2DArraySRF, ChangeCoordinate2DArraySRFObject, ChangeCoordinate2DSRF, ChangeCoordinate2DSRFObject, CreateCoordinate2D, EuclideanDistance, GetCoordinate2DValues, InRTRegionTest
Method	Create
Semantics	Overrides the Create method of the superclass LifeCycleObject. Creates a LocalSpaceRectangular2D SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: LSR_2D_Parameters
Outputs	new_srf: LocalSpaceRectangular2D
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LOCAL_SPACE_RECTANGULAR_2D, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid LSR_2D_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: LSR_2D_Parameters
Error conditions	No additional error conditions.

11.3.7 SRF concrete subclasses of BaseSRF3D

11.3.7.1 Introduction

The direct concrete subclasses of `BaseSRF3D` are:

`Celestiocentric`,
`Celestiomagnetic`,
`EquatorialInertial`,
`HeliosphericAriesEcliptic`,
`HeliosphericEarthEcliptic`,
`HeliosphericEarthEquatorial`,
`LocalSpaceRectangular3D`,
`LococentricEuclidean3D`,
`SolarEcliptic`,
`SolarEquatorial`,
`SolarMagneticDipole`, and
`SolarMagneticEcliptic`.

These concrete classes override the `Create` method of `LifeCycleObject` so that an instance of the class can be created. In those cases for which the `Create` method requires additional SRF-specific parameters, a `GetSRFParameters` method is also specified for the class.

11.3.7.2 Celestiocentric

Table 11.21 — Celestiocentric

Element	Specification
Class	`Celestiocentric`
Description	An instance of this class corresponds to an instance of SRFT CELESTIOCENTRIC.

Superclass(es)	`LifeCycleObject`: `Destroy`
	`BaseSRF`: `GetCSCode`, `GetORMCodes`, `GetSRFCodes`
	`BaseSRF3D`: `ChangeCoordinate3DArraySRF`, `ChangeCoordinate3DArraySRFObject`, `ChangeCoordinate3DSRF`, `ChangeCoordinate3DSRFObject`, `ChangeDirectionArraySRF`, `ChangeDirectionArraySRFObject`, `ChangeDirectionSRF`, `ChangeDirectionSRFObject`, `ComputeSRFOrientation`, `CreateCoordinate3D`, `CreateDirection`, `CreateLococentricEuclidean3DSRF`, `EuclideanDistance`, `GetCoordinate3DValues`, `GetDirectionValues`, `GetLocalTangentFrameSRFParameters`, `GetSRFRegion`, `InRTRegionTest`, `InSRFRegionTest`, `InSRFRegionTestArray`, `SetSRFRegion`, `TransformOrientation`, `TransformOrientationCommonOrigin`, `TransformVector`, `TransformVectorCommonOrigin`, `TransformVectorInBodyFrame`, `TransformVectorInBodyFrameCommonOrigin`

Method	`Create`

Element	Specification
Class	`Celestiocentric`
Semantics	Overrides the `Create` method on the superclass `LifeCycleObject`. Creates a `Celestiocentric` SRF corresponding to the input values. The input `rt_code` value 0 (`UNSPECIFIED`) is permitted. However, if 0 (`UNSPECIFIED`) is used, methods that perform spatial operations involving another input SRF will produce an error condition (`OPERATION_UNSUPPORTED`), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	`orm_code:` `ORM_Code` `rt_code:` `RT_Code`
Outputs	`new_srf:` `Celestiocentric`
Error conditions	1) `UNDEFINED_CODE` if the value of `orm_code` or `rt_code` is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) `INVALID_CODE` if `orm_code` has the `ORM_Code` value 0 (`UNSPECIFIED`). 3) `INCOMPATIBLE_CODE` if (1) the ORM specified by `orm_code` is not compatible with the ORM constraints of SRFT CELESTIOCENTRIC, or (2) the value of `rt_code` is not compatible with the value of `orm_code`.

11.3.7.3 Celestiomagnetic

Table 11.22 — Celestiomagnetic

Element	Specification
Class	`Celestiomagnetic`
Description	An instance of this class corresponds to an instance of SRFT CELESTIOMAGNETIC.

Superclass(es)	`LifeCycleObject`: `Destroy`
	`BaseSRF`: `GetCSCode`, `GetORMCodes`, `GetSRFCodes`
	`BaseSRF3D`: `ChangeCoordinate3DArraySRF`, `ChangeCoordinate3DArraySRFObject`, `ChangeCoordinate3DSRF`, `ChangeCoordinate3DSRFObject`, `ChangeDirectionArraySRF`, `ChangeDirectionArraySRFObject`, `ChangeDirectionSRF`, `ChangeDirectionSRFObject`, `ComputeSRFOrientation`, `CreateCoordinate3D`, `CreateDirection`, `CreateLococentricEuclidean3DSRF`, `EuclideanDistance`, `GetCoordinate3DValues`, `GetDirectionValues`, `GetLocalTangentFrameSRFParameters`, `GetSRFRegion`, `InRTRegionTest`, `InSRFRegionTest`, `InSRFRegionTestArray`, `SetSRFRegion`, `TransformOrientation`, `TransformOrientationCommonOrigin`, `TransformVector`, `TransformVectorCommonOrigin`, `TransformVectorInBodyFrame`, `TransformVectorInBodyFrameCommonOrigin`

Method	`Create`
Semantics	Overrides the `Create` method on the superclass `LifeCycleObject`. Creates a `Celestiomagnetic` SRF corresponding to the input values. The input `rt_code` value 0 (`UNSPECIFIED`) is permitted. However, if 0 (`UNSPECIFIED`) is used, methods that perform spatial operations involving another input SRF will produce an error condition (`OPERATION_UNSUPPORTED`), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.

Element	Specification
Class	Celestiomagnetic
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: Celestiomagnetic
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT CELESTIOMAGNETIC, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.7.4 EquatorialInertial

Table 11.23 — EquatorialInertial

Element	Specification
Class	EquatorialInertial
Description	An instance of this class corresponds to an instance of SRFT EQUATORIAL_INERTIAL.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates an EquatorialInertial SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: EquatorialInertial

Element	Specification
Class	EquatorialInertial
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT EQUATORIAL_INERTIAL, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.7.5 HeliosphericAriesEcliptic

Table 11.24 — HeliosphericAriesEcliptic

Element	Specification
Class	HeliosphericAriesEcliptic
Description	An instance of this class corresponds to an instance of SRFT HELIOSPHERIC_ARIES_ECLIPTIC.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a HeliosphericAriesEcliptic SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: HeliosphericAriesEcliptic
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT HELIOSPHERIC_ARIES_ECLIPTIC, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.7.6 HeliosphericEarthEcliptic

Table 11.25 — HeliosphericEarthEcliptic

Element	Specification
Class	`HeliosphericEarthEcliptic`
Description	An instance of this class corresponds to an instance of SRFT HELIOSPHERIC_EARTH_ECLIPTIC.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	`Create`
Semantics	Overrides the `Create` method on the superclass LifeCycleObject. Creates a `HeliosphericEarthEcliptic` SRF corresponding to the input values. The input `rt_code` value 0 (`UNSPECIFIED`) is permitted. However, if 0 (`UNSPECIFIED`) is used, methods that perform spatial operations involving another input SRF will produce an error condition (`OPERATION_UNSUPPORTED`), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	`orm_code:` ORM_Code `rt_code:` RT_Code
Outputs	`new_srf:` HeliosphericEarthEcliptic
Error conditions	1) `UNDEFINED_CODE` if the value of `orm_code` or `rt_code` is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) `INVALID_CODE` if `orm_code` has the `ORM_Code` value 0 (`UNSPECIFIED`). 3) `INCOMPATIBLE_CODE` if (1) the ORM specified by `orm_code` is not compatible with the ORM constraints of SRFT HELIOSPHERIC_EARTH_ECLIPTIC, or (2) the value of `rt_code` is not compatible with the value of `orm_code`.

11.3.7.7 HeliosphericEarthEquatorial

Table 11.26 — HeliosphericEarthEquatorial

Element	Specification
Class	`HeliosphericEarthEquatorial`
Description	An instance of this class corresponds to an instance of SRFT HELIOSPHERIC_EARTH_EQUATORIAL.

Element	Specification
Class	HeliosphericEarthEquatorial
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a HeliosphericEarthEquatorial SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: HeliosphericEarthEquatorial
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT HELIOSPHERIC_EARTH_EQUATORIAL, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.7.8 LocalSpaceRectangular3D

Table 11.27 — LocalSpaceRectangular3D

Element	Specification
Class	LocalSpaceRectangular3D
Description	An instance of this class corresponds to an instance of SRFT LOCAL_SPACE_RECTANGULAR_3D.

Element	Specification
Class	LocalSpaceRectangular3D
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a LocalSpaceRectangular3D SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: LSR_3D_Parameters
Outputs	new_srf: LocalSpaceRectangular3D
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LOCAL_SPACE_RECTANGULAR_3D, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid LSR_3D_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: LSR_3D_Parameters
Error conditions	No additional error conditions.

11.3.7.9 LococentricEuclidean3D

Table 11.28 — LococentricEuclidean3D

Element	Specification
Class	LococentricEuclidean3D
Description	An instance of this class corresponds to an instance of SRFT LOCOCENTRIC_EUCLIDEAN_3D.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a LococentricEuclidean3D SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: LCE_3D_Parameters
Outputs	new_srf: LococentricEuclidean3D
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LOCOCENTRIC_EUCLIDEAN_3D, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid LCE_3D_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: LCE_3D_Parameters
Error conditions	No additional error conditions.

11.3.7.10 SolarEcliptic

Table 11.29 — SolarEcliptic

Element	Specification
Class	SolarEcliptic
Description	An instance of this class corresponds to an instance of SRFT SOLAR_ECLIPTIC.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a SolarEcliptic SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: SolarEcliptic
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT SOLAR_ECLIPTIC, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.7.11 SolarEquatorial

Table 11.30 — SolarEquatorial

Element	Specification
Class	SolarEquatorial
Description	An instance of this class corresponds to an instance of SRFT SOLAR_EQUATORIAL.

Element	Specification
Class	SolarEquatorial
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a SolarEquatorial SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: SolarEquatorial
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT SOLAR_EQUATORIAL, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.7.12 SolarMagneticDipole

Table 11.31 — SolarMagneticDipole

Element	Specification
Class	SolarMagneticDipole
Description	An instance of this class corresponds to an instance of SRFT SOLAR_MAGNETIC_DIPOLE.

Element	Specification
Class	SolarMagneticDipole
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a SolarMagneticDipole SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: SolarMagneticDipole
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT SOLAR_MAGNETIC_DIPOLE, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.7.13 SolarMagneticEcliptic

Table 11.32 — SolarMagneticEcliptic

Element	Specification
Class	SolarMagneticEcliptic
Description	An instance of this class corresponds to an instance of SRFT SOLAR_MAGNETIC_ECLIPTIC.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a SolarMagneticEcliptic SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	new_srf: SolarMagneticEcliptic
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT SOLAR_MAGNETIC_ECLIPTIC, or (2) the value of rt_code is not compatible with the value of orm_code.

11.3.8 SRF concrete subclasses of BaseSRFMapProjection

11.3.8.1 Introduction

The concrete subclasses of BaseSRFMapProjection are:

```
EquidistantCylindrical,
LambertConformalConic,
Mercator,
ObliqueMercatorSpherical,
```

PolarStereographic, and
TransverseMercator.

These concrete classes override the Create method of LifeCycleObject so that an instance of the class can be created. In those cases for which the Create method requires additional SRF-specific parameters, a GetSRFParameters method is also specified for the class.

11.3.8.2 EquidistantCylindrical

Table 11.33 — EquidistantCylindrical

Element	Specification
Class	EquidistantCylindrical
Description	An instance of this class corresponds to an instance of SRFT EQUIDISTANT_CYLINDRICAL.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithEllipsoidalHeight: CreateLocalTangentSpaceEuclideanSRF, CreateSurfaceCoordinate, EuclideanDistance, GeodesicDestination, GeodesicDistance, GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues, InRTRegionTest, PromoteSurfaceCoordinate, TruncateCoordinate3D, VerticalOffset BaseSRFMapProjection: ConvergenceOfTheMeridian, MapAzimuth, PointDistortion
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates an EquidistantCylindrical SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: EC_Parameters
Outputs	new_srf: EquidistantCylindrical

Element	Specification
Class	EquidistantCylindrical
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT EQUIDISTANT_CYLINDRICAL, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid EC_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: EC_Parameters
Error conditions	No additional error conditions.

11.3.8.3 LambertConformalConic

Table 11.34 — LambertConformalConic

Element	Specification
Class	LambertConformalConic
Description	An instance of this class corresponds to an instance of SRFT LAMBERT_CONFORMAL_CONIC.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithEllipsoidalHeight: CreateLocalTangentSpaceEuclideanSRF, CreateSurfaceCoordinate, EuclideanDistance, GeodesicDestination, GeodesicDistance, GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues, InRTRegionTest, PromoteSurfaceCoordinate, TruncateCoordinate3D, VerticalOffset BaseSRFMapProjection: ConvergenceOfTheMeridian, MapAzimuth, PointDistortion
Method	Create

Element	Specification
Class	LambertConformalConic
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a LambertConformalConic SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: LCC_Parameters
Outputs	new_srf: LambertConformalConic
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LAMBERT_CONFORMAL_CONIC, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid LCC_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: LCC_Parameters
Error conditions	No additional error conditions.

11.3.8.4 Mercator

Table 11.35 — Mercator

Element	**Specification**
Class	Mercator
Description	An instance of this class corresponds to an instance of SRFT MERCATOR.

Element	Specification
Class	Mercator
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithEllipsoidalHeight: CreateLocalTangentSpaceEuclideanSRF, CreateSurfaceCoordinate, EuclideanDistance, GeodesicDestination, GeodesicDistance, GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues, InRTRegionTest, PromoteSurfaceCoordinate, TruncateCoordinate3D, VerticalOffset BaseSRFMapProjection: ConvergenceOfTheMeridian, MapAzimuth, PointDistortion
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a Mercator SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: M_Parameters
Outputs	new_srf: Mercator
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT MERCATOR, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid M_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: M_Parameters

Element	**Specification**
Class	Mercator
Error conditions	No additional error conditions.

11.3.8.5 ObliqueMercatorSpherical

Table 11.36 — ObliqueMercatorSpherical

Element	**Specification**
Class	ObliqueMercatorSpherical
Description	An instance of this class corresponds to an instance of SRFT OBLIQUE_MERCATOR_SPHERICAL.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithEllipsoidalHeight: CreateLocalTangentSpaceEuclideanSRF, CreateSurfaceCoordinate, EuclideanDistance, GeodesicDestination, GeodesicDistance, GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues, InRTRegionTest, PromoteSurfaceCoordinate, TruncateCoordinate3D, VerticalOffset BaseSRFMapProjection: ConvergenceOfTheMeridian, MapAzimuth, PointDistortion
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates an ObliqueMercatorSpherical SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: Oblique_Mercator_Parameters
Outputs	new_srf: ObliqueMercatorSpherical

Element	Specification
Class	ObliqueMercatorSpherical
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT OBLIQUE_MERCATOR_SPHERICAL, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid Oblique_Mercator_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: Oblique_Mercator_Parameters
Error conditions	No additional error conditions.

11.3.8.6 PolarStereographic

Table 11.37 — PolarStereographic

Element	Specification
Class	PolarStereographic
Description	An instance of this class corresponds to an instance of SRFT POLAR_STEREOGRAPHIC.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithEllipsoidalHeight: CreateLocalTangentSpaceEuclideanSRF, CreateSurfaceCoordinate, EuclideanDistance, GeodesicDestination, GeodesicDistance, GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues, InRTRegionTest, PromoteSurfaceCoordinate, TruncateCoordinate3D, VerticalOffset BaseSRFMapProjection: ConvergenceOfTheMeridian, MapAzimuth, PointDistortion
Method	Create

Element	Specification
Class	PolarStereographic
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a PolarStereographic SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: PS_Parameters
Outputs	new_srf: PolarStereographic
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT POLAR_STEREOGRAPHIC, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid PS_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: PS_Parameters
Error conditions	No additional error conditions.

11.3.8.7 TransverseMercator

Table 11.38 — TransverseMercator

Element	Specification
Class	TransverseMercator
Description	An instance of this class corresponds to an instance of SRFT TRANSVERSE_MERCATOR.

Element	Specification
Class	TransverseMercator
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithEllipsoidalHeight: CreateLocalTangentSpaceEuclideanSRF, CreateSurfaceCoordinate, EuclideanDistance, GeodesicDestination, GeodesicDistance, GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues, InRTRegionTest, PromoteSurfaceCoordinate, TruncateCoordinate3D, VerticalOffset BaseSRFMapProjection: ConvergenceOfTheMeridian, MapAzimuth, PointDistortion
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a TransverseMercator SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: TM_Parameters
Outputs	new_srf: TransverseMercator
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT TRANSVERSE_MERCATOR, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid TM_Parameters data structure.

Element	Specification
Class	`TransverseMercator`
Method	`GetSRFParameters`
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	`parameters:` `TM_Parameters`
Error conditions	No additional error conditions.

11.3.9 SRF concrete subclasses of BaseSRFwithEllipsoidalHeight

11.3.9.1 Introduction

The direct concrete subclasses of `BaseSRFwithEllipsoidalHeight` are `Celestiodetic` and `Planetodetic`. These concrete classes override the `Create` method of `LifeCycleObject` so that an instance of the class can be created. In those cases for which the `Create` method requires additional SRF-specific parameters, a `GetSRFParameters` method is also specified for the class.

11.3.9.2 Celestiodetic

Table 11.39 — Celestiodetic

Element	Specification
Class	`Celestiodetic`
Description	An instance of this class corresponds to an instance of SRFT CELESTIODETIC.

Superclass(es)	`LifeCycleObject: Destroy`
	`BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes`
	`BaseSRF3D: ChangeCoordinate3DArraySRF,`
	`ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF,`
	`ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF,`
	`ChangeDirectionArraySRFObject, ChangeDirectionSRF,`
	`ChangeDirectionSRFObject, ComputeSRFOrientation,`
	`CreateCoordinate3D, CreateDirection,`
	`CreateLococentricEuclidean3DSRF, EuclideanDistance,`
	`GetCoordinate3DValues, GetDirectionValues,`
	`GetLocalTangentFrameSRFParameters, GetSRFRegion,`
	`InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray,`
	`SetSRFRegion, TransformOrientation,`
	`TransformOrientationCommonOrigin, TransformVector,`
	`TransformVectorCommonOrigin, TransformVectorInBodyFrame,`
	`TransformVectorInBodyFrameCommonOrigin`
	`BaseSRFwithEllipsoidalHeight:`
	`CreateLocalTangentSpaceEuclideanSRF,`
	`CreateSurfaceCoordinate, EuclideanDistance,`
	`GeodesicDestination, GeodesicDistance,`
	`GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues,`
	`InRTRegionTest, PromoteSurfaceCoordinate,`
	`TruncateCoordinate3D, VerticalOffset`

Method	`Create`
Semantics	Overrides the `Create` method on the superclass `LifeCycleObject`. Creates a `Celestiodetic` SRF corresponding to the input values. The input `rt_code` value 0 (`UNSPECIFIED`) is permitted. However, if 0 (`UNSPECIFIED`) is used, methods that perform spatial operations involving another input SRF will produce an error condition (`OPERATION_UNSUPPORTED`), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	`orm_code:` `ORM_Code` `rt_code:` `RT_Code`
Outputs	`new_srf:` `Celestiodetic`
Error conditions	1) `UNDEFINED_CODE` if the value of `orm_code` or `rt_code` is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) `INVALID_CODE` if `orm_code` has the `ORM_Code` value 0 (`UNSPECIFIED`). 3) `INCOMPATIBLE_CODE` if (1) the ORM specified by `orm_code` is not compatible with the ORM constraints of SRFT CELESTIODETIC, or (2) the value of `rt_code` is not compatible with the value of `orm_code`.

11.3.9.3 Planetodetic

Table 11.40 — Planetodetic

Element	**Specification**
Class	`Planetodetic`
Description	An instance of this class corresponds to an instance of SRFT PLANETODETIC.

Superclass(es)	`LifeCycleObject: Destroy`
	`BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes`
	`BaseSRF3D: ChangeCoordinate3DArraySRF,`
	`ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF,`
	`ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF,`
	`ChangeDirectionArraySRFObject, ChangeDirectionSRF,`
	`ChangeDirectionSRFObject, ComputeSRFOrientation,`
	`CreateCoordinate3D, CreateDirection,`
	`CreateLococentricEuclidean3DSRF, EuclideanDistance,`
	`GetCoordinate3DValues, GetDirectionValues,`
	`GetLocalTangentFrameSRFParameters, GetSRFRegion,`
	`InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray,`
	`SetSRFRegion, TransformOrientation,`
	`TransformOrientationCommonOrigin, TransformVector,`
	`TransformVectorCommonOrigin, TransformVectorInBodyFrame,`
	`TransformVectorInBodyFrameCommonOrigin`
	`BaseSRFwithEllipsoidalHeight:`
	`CreateLocalTangentSpaceEuclideanSRF,`
	`CreateSurfaceCoordinate, EuclideanDistance,`
	`GeodesicDestination, GeodesicDistance,`
	`GeodesicDistanceWithAzimuths, GetSurfaceCoordinateValues,`
	`InRTRegionTest, PromoteSurfaceCoordinate,`
	`TruncateCoordinate3D, VerticalOffset`

Method	`Create`
Semantics	Overrides the `Create` method on the superclass `LifeCycleObject`. Creates a `Planetodetic` SRF corresponding to the input values. The input `rt_code` value 0 (`UNSPECIFIED`) is permitted. However, if 0 (`UNSPECIFIED`) is used, methods that perform spatial operations involving another input SRF will produce an error condition (`OPERATION_UNSUPPORTED`), if the method does not also require an $\boldsymbol{H_{ST}}$ input.
Inputs	`orm_code:` `ORM_Code` `rt_code:` `RT_Code`
Outputs	`new_srf:` `Planetodetic`
Error conditions	1) `UNDEFINED_CODE` if the value of `orm_code` or `rt_code` is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) `INVALID_CODE` if `orm_code` has the `ORM_Code` value 0 (`UNSPECIFIED`). 3) `INCOMPATIBLE_CODE` if (1) the ORM specified by `orm_code` is not compatible with the ORM constraints of SRFT PLANETODETIC, or (2) the value of `rt_code` is not compatible with the value of `orm_code`.

11.3.10 SRF concrete subclasses of BaseSRFwithTangentPlaneSurface

11.3.10.1 Introduction

The concrete subclasses of `BaseSRFwithTangentPlaneSurface` are:

LocalTangentSpaceAzimuthalSpherical,
LocalTangentSpaceCylindrical, and
LocalTangentSpaceEuclidean.

These concrete classes override the Create method of LifeCycleObject so that an instance of the class can be created. In those cases for which the Create method requires additional SRF-specific parameters, a GetSRFParameters method is also specified for the class.

11.3.10.2 LocalTangentSpaceAzimuthalSpherical

Table 11.41 — LocalTangentSpaceAzimuthalSpherical

Element	Specification
Class	LocalTangentSpaceAzimuthalSpherical
Description	An instance of this class corresponds to an instance of SRFT LOCAL_TANGENT_SPACE_AZIMUTHAL_SPHERICAL.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithTangentPlaneSurface: CreateSurfaceCoordinate, EuclideanDistance, GetSurfaceCoordinateValues, PromoteSurfaceCoordinate, TruncateCoordinate3D
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a LocalTangentSpaceAzimuthalSpherical SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an H_{ST} input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: Local_Tangent_Parameters
Outputs	new_srf: LocalTangentSpaceAzimuthalSpherical

Element	Specification
Class	LocalTangentSpaceAzimuthalSpherical
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LOCAL_TANGENT_SPACE_AZIMUTHAL_SPHERICAL, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid Local_Tangent_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: Local_Tangent_Parameters
Error conditions	No additional error conditions.

11.3.10.3 LocalTangentSpaceCylindrical

Table 11.42 — LocalTangentSpaceCylindrical

Element	Specification
Class	LocalTangentSpaceCylindrical
Description	An instance of this class corresponds to an instance of SRFT LOCAL_TANGENT_SPACE_CYLINDRICAL.
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithTangentPlaneSurface: CreateSurfaceCoordinate, EuclideanDistance, GetSurfaceCoordinateValues, PromoteSurfaceCoordinate, TruncateCoordinate3D

Element	Specification
Class	LocalTangentSpaceCylindrical
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a LocalTangentSpaceCylindrical SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: Local_Tangent_Parameters
Outputs	new_srf: LocalTangentSpaceCylindrical
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LOCAL_TANGENT_SPACE_CYLINDRICAL, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid Local_Tangent_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: Local_Tangent_Parameters
Error conditions	No additional error conditions.

11.3.10.4 LocalTangentSpaceEuclidean

Table 11.43 — LocalTangentSpaceEuclidean

Element	Specification
Class	LocalTangentSpaceEuclidean
Description	An instance of this class corresponds to an instance of SRFT LOCAL_TANGENT_SPACE_EUCLIDEAN.

Element	Specification
Class	LocalTangentSpaceEuclidean
Superclass(es)	LifeCycleObject: Destroy BaseSRF: GetCSCode, GetORMCodes, GetSRFCodes BaseSRF3D: ChangeCoordinate3DArraySRF, ChangeCoordinate3DArraySRFObject, ChangeCoordinate3DSRF, ChangeCoordinate3DSRFObject, ChangeDirectionArraySRF, ChangeDirectionArraySRFObject, ChangeDirectionSRF, ChangeDirectionSRFObject, ComputeSRFOrientation, CreateCoordinate3D, CreateDirection, CreateLococentricEuclidean3DSRF, EuclideanDistance, GetCoordinate3DValues, GetDirectionValues, GetLocalTangentFrameSRFParameters, GetSRFRegion, InRTRegionTest, InSRFRegionTest, InSRFRegionTestArray, SetSRFRegion, TransformOrientation, TransformOrientationCommonOrigin, TransformVector, TransformVectorCommonOrigin, TransformVectorInBodyFrame, TransformVectorInBodyFrameCommonOrigin BaseSRFwithTangentPlaneSurface: CreateSurfaceCoordinate, EuclideanDistance, GetSurfaceCoordinateValues, PromoteSurfaceCoordinate, TruncateCoordinate3D
Method	Create
Semantics	Overrides the Create method on the superclass LifeCycleObject. Creates a LocalTangentEuclidean SRF corresponding to the input values. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.
Inputs	orm_code: ORM_Code rt_code: RT_Code parameters: LTSE_Parameters
Outputs	new_srf: LocalTangentSpaceEuclidean
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if (1) the ORM specified by orm_code is not compatible with the ORM constraints of SRFT LOCAL_TANGENT_SPACE_EUCLIDEAN, or (2) the value of rt_code is not compatible with the value of orm_code. 4) INVALID_PARAMETERS if the input parameters is not a valid LTSE_Parameters data structure.
Method	GetSRFParameters
Semantics	Outputs the SRF parameter values.
Inputs	none
Outputs	parameters: LTSE_Parameters
Error conditions	No additional error conditions.

11.3.11 Concrete subclasses of Orientation

11.3.11.1 Introduction

The concrete subclasses of Orientation are:

`OrientationAxisAngle`,
`OrientationEulerAnglesZXZ`,
`OrientationMatrix`,
`OrientationQuaternion`,
`OrientationTaitBryanAnglesXYZ`, and
`OrientationTaitBryanAnglesZYX`.

These concrete classes override the `Create` method of `LifeCycleObject` so that an instance of the class can be created. In those cases for which the `Create` method requires additional orientation-specific parameters, `Set` and `Get` methods are also specified for the class.

11.3.11.2 OrientationAxisAngle

Table 11.44 — OrientationAxisAngle

Element	Specification
Class	`OrientationAxisAngle`
Description	An instance of this class corresponds to an orientation using an axis-angle representation.
Superclass(es)	`LifeCycleObject`: `Destroy` `Orientation`: `ComposeWith, GetAxisAngle, GetEulerAnglesZXZ, GetMatrix3x3, GetQuaternion, GetTaitBryanAnglesXYZ, GetTaitBryanAnglesZYX, TransformVector`
Method	`Create`
Semantics	Creates an `OrientationAxisAngle` instance according to the input values.
Inputs	`axis: Vector_3D` `angle: Long_Float`
Outputs	`new_orientation: OrientationAxisAngle`
Error conditions	No additional error conditions.
Method	`Get`
Semantics	Outputs the parameter values of this `OrientationAxisAngle` instance.
Inputs	none
Outputs	`parameters: Axis_Angle_Parameters`
Error conditions	No additional error conditions.
Method	`Set`
Semantics	Sets the parameters of this `OrientationAxisAngle` instance.
Inputs	`parameters: Axis_Angle_Parameters`
Outputs	none
Error conditions	`INVALID_PARAMETERS` if the input `parameters` is not a valid `Axis_Angle_Parameters` data structure.

11.3.11.3 OrientationEulerAnglesZXZ

Table 11.45 — OrientationEulerAnglesZXZ

Element	Specification
Class	`OrientationEulerAnglesZXZ`
Description	An instance of this class corresponds to an orientation using Euler angle ZXZ rotation representation: $\boldsymbol{R}_{z''}(\text{precession}) \circ \boldsymbol{R}_{x'}(\text{nutation}) \circ \boldsymbol{R}_{z}(\text{spin})$ $= \boldsymbol{R}_{z}(\text{spin}) \circ \boldsymbol{R}_{x}(\text{nutation}) \circ \boldsymbol{R}_{z}(\text{precession}).$
Superclass(es)	`LifeCycleObject`: `Destroy` `Orientation`: `ComposeWith, GetAxisAngle, GetEulerAnglesZXZ, GetMatrix3x3, GetQuaternion, GetTaitBryanAnglesXYZ, GetTaitBryanAnglesZYX, TransformVector`
Method	`Create`
Semantics	Creates an `OrientationEulerAnglesZXZ` instance according to the input values.

Inputs	`spin: Long_Float`
	`nutation: Long_Float`
	`precession: Long_Float`

Outputs	`new_orientation: OrientationEulerAnglesZXZ`
Error conditions	No additional error conditions.
Method	`Get`
Semantics	Outputs the parameters of this `OrientationEulerAnglesZXZ` instance.
Inputs	none
Outputs	`parameters: Euler_Angles_ZXZ_Parameters`
Error conditions	No additional error conditions.
Method	`Set`
Semantics	Sets the parameters of this `OrientationEulerAnglesZXZ` instance.
Inputs	`parameters: Euler_Angles_ZXZ_Parameters`
Outputs	none
Error conditions	`INVALID_PARAMETERS` if the input `parameters` is not a valid data `Euler_Angles_ZXZ_Parameters` structure.

11.3.11.4 OrientationMatrix

Table 11.46 — OrientationMatrix

Element	Specification
Class	`OrientationMatrix`
Description	An instance of this class corresponds to an orientation with a 3x3 rotation matrix representation.
Superclass(es)	`LifeCycleObject`: `Destroy` `Orientation`: `ComposeWith, GetAxisAngle, GetEulerAnglesZXZ, GetMatrix3x3, GetQuaternion, GetTaitBryanAnglesXYZ, GetTaitBryanAnglesZYX, TransformVector`

Element	Specification
Class	`OrientationMatrix`
Method	`Create`
Semantics	Creates an `OrientationMatrix` instance according to the input values.
Inputs	`m:` `Matrix_3x3`
Outputs	`new_orientation:` `OrientationMatrix`
Error conditions	No additional error conditions.
Method	`Get`
Semantics	Outputs the parameters of this `OrientationMatrix` instance.
Inputs	none
Outputs	`parameters:` `Matrix_3x3`
Error conditions	No additional error conditions.
Method	`Set`
Semantics	Sets the parameters of this `OrientationMatrix` instance.
Inputs	`parameters:` `Matrix_3x3`
Outputs	none
Error conditions	`INVALID_PARAMETERS` if the input `parameters` is not a valid `Matrix_3x3` data structure.

11.3.11.5 OrientationQuaternion

Table 11.47 — OrientationQuaternion

Element	Specification
Class	`OrientationQuaternion`
Description	An instance of this class corresponds to an orientation with a quaternion representation.
Superclass(es)	`LifeCycleObject`: `Destroy` `Orientation`: `ComposeWith, GetAxisAngle, GetEulerAnglesZXZ, GetMatrix3x3, GetQuaternion, GetTaitBryanAnglesXYZ, GetTaitBryanAnglesZYX, TransformVector`
Method	`Create`
Semantics	Creates an `OrientationQuaternion` instance according to the input values.

Inputs	`e0:` `Long_Float`
	`e1:` `Long_Float`
	`e2:` `Long_Float`
	`e3:` `Long_Float`

Outputs	`new_orientation:` `OrientationQuaternion`
Error conditions	No additional error conditions.
Method	`Get`
Semantics	Outputs the parameters of this `OrientationQuaternion` instance.
Inputs	none

Element	Specification
Class	`OrientationQuaternion`
Outputs	`parameters:` `Quaternion_Parameters`
Error conditions	No additional error conditions.
Method	`Set`
Semantics	Sets the parameters of this `OrientationQuaternion` instance.
Inputs	`parameters:` `Quaternion_Parameters`
Outputs	none
Error conditions	`INVALID_PARAMETERS` if the input `parameters` is not a valid `Quaternion_Parameters` data structure.

11.3.11.6 OrientationTaitBryanAnglesXYZ

Table 11.48 — OrientationTaitBryanAnglesXYZ

Element	Specification
Class	`OrientationTaitBryanAnglesXYZ`
Description	An instance of this class corresponds to an orientation with Tait-Bryan angles XYZ rotation representation: $\boldsymbol{R}_{z''}(\text{yaw}) \circ \boldsymbol{R}_{y'}(\text{pitch}) \circ \boldsymbol{R}_{x}(\text{roll})$ $= \boldsymbol{R}_{x}(\text{roll}) \circ \boldsymbol{R}_{y}(\text{pitch}) \circ \boldsymbol{R}_{z}(\text{yaw})$.
Superclass(es)	`LifeCycleObject`: `Destroy` `Orientation`: `ComposeWith, GetAxisAngle, GetEulerAnglesZXZ, GetMatrix3x3, GetQuaternion, GetTaitBryanAnglesXYZ, GetTaitBryanAnglesZYX, TransformVector`
Method	`Create`
Semantics	Creates an `OrientationTaitBryanAnglesXYZ` instance according to the input values.

Inputs	`roll:` `Long_Float`
	`pitch:` `Long_Float`
	`yaw:` `Long_Float`

Outputs	`new_orientation:` `OrientationTaitBryanAnglesXYZ`
Error conditions	No additional error conditions.
Method	`Get`
Semantics	Outputs the parameters of this `OrientationTaitBryanAnglesXYZ` instance.
Inputs	none
Outputs	`parameters:` `Tait_Bryan_Parameters`
Error conditions	No additional error conditions.
Method	`Set`
Semantics	Sets the parameters of this `OrientationTaitBryanAnglesXYZ` instance.
Inputs	`parameters:` `Tait_Bryan_Parameters`
Outputs	none

Element	Specification
Class	`OrientationTaitBryanAnglesXYZ`
Error conditions	`INVALID_PARAMETERS` if the input `parameters` is not a valid `Tait_Bryan_Parameters` data structure.

11.3.11.7 OrientationTaitBryanAnglesZYX

Table 11.49 — OrientationTaitBryanAnglesZYX

Element	Specification
Class	`OrientationTaitBryanAnglesZYX`
Description	An instance of this class corresponds to an orientation with Tait-Bryan angles ZYX rotation representation: $\boldsymbol{R}_{x''}(\text{roll}) \circ \boldsymbol{R}_{y'}(\text{pitch}) \circ \boldsymbol{R}_{z}(\text{yaw}) = \boldsymbol{R}_{z}(\text{yaw}) \circ \boldsymbol{R}_{y}(\text{pitch}) \circ \boldsymbol{R}_{x}(\text{roll})$.
Superclass(es)	`LifeCycleObject`: `Destroy` `Orientation`: `ComposeWith, GetAxisAngle, GetEulerAnglesZXZ, GetMatrix3x3, GetQuaternion, GetTaitBryanAnglesXYZ, GetTaitBryanAnglesZYX, TransformVector`
Method	`Create`
Semantics	Creates an `OrientationTaitBryanAnglesZYX` instance according to the input values.

Inputs	`roll: Long_Float`
	`pitch: Long_Float`
	`yaw: Long_Float`

Outputs	`new_orientation: OrientationTaitBryanAnglesZYX`
Error conditions	No additional error conditions.
Method	`Get`
Semantics	Outputs the parameters of this `OrientationTaitBryanAnglesZYX` instance.
Inputs	none
Outputs	`parameters: Tait_Bryan_Parameters`
Error conditions	No additional error conditions.
Method	`Set`
Semantics	Sets the parameters of this `OrientationTaitBryanAnglesZYX` instance.
Inputs	`parameters: Tait_Bryan_Parameters`
Outputs	none
Error conditions	`INVALID_PARAMETERS` if the input `parameters` is not a valid `Tait_Bryan_Parameters` data structure.

11.4 Functions

11.4.1 Function to create instances of standardized SRFs

This subclause defines the `CreateStandardizedSRF` function (see Table 11.50) that creates a concrete SRF class instance corresponding to either one of the standardized SRFs in Table 8.31 or a registered SRF (see

13.3.10). When the GetSRFCodes method is invoked on an instance of an SRF created using this function, the corresponding SRF_Code is returned. However, when the GetSRFCodes method is invoked on an instance of an SRF created using the Create method of a concrete SRF class, the SRF_Code 0 (UNSPECIFIED) is returned.

Table 11.50 — CreateStandardizedSRF

Element	Specification
Function	CreateStandardizedSRF
Semantics	Creates an SRF instance corresponding to either one of the standardized SRFs in Table 8.31 or a registered SRF (see 13.3.10). The specific SRF is specified by srf_code. The input rt_code value 0 (UNSPECIFIED) is permitted. However, if 0 (UNSPECIFIED) is used, methods that perform spatial operations involving another input SRF will produce an error condition (OPERATION_UNSUPPORTED), if the method does not also require an $\boldsymbol{H}_{ST}$ input.
Inputs	srf_code: SRF_Code rt_code: RT_Code
Outputs	new_srf: (concrete subclass of) BaseSRF
Error conditions	1) UNDEFINED_CODE if the value of srf_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if srf_code has the SRF_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if the value of rt_code is not compatible with the ORM identified by the value of srf_code.

See also common error conditions in 11.2.7.12.

EXAMPLE CreateStandardizedSRF with srf_code = 4 and an rt_code, produces as output a Celestiocentric instance corresponding to SRF GEOCENTRIC_WGS_1984.

11.4.2 Function to create instances of standardized SRF set members

This subclause defines the CreateStandardizedSRFSetMember function (see Table 11.51) that creates a concrete SRF class instance corresponding to one of the set members for either one of the standardized SRF sets in Table 8.48 or a registered SRF set (see 13.3.11). When the GetSRFCodes method is invoked on an instance of an SRF created using this function, the corresponding SRF set member information is returned in SRFS_Code_Info. However, when the GetSRFCodes method is invoked on an instance of an SRF created using the Create method of a concrete SRF class, the SRFS_Code 0 (UNSPECIFIED) is returned in the SRFS_Code_Info structure.

Table 11.51 — CreateStandardizedSRFSetMember

Element	Specification
Function	`CreateStandardizedSRFSetMember`
Semantics	Creates an SRF instance corresponding to one of the set members for either one of the standardized SRF sets in Table 8.48 or a registered SRF set (see 13.3.11). The specific SRF set and its member is specified by the `srfs_code_info` and `orm_code` inputs. The input `rt_code` value 0 (`UNSPECIFIED`) is permitted. However, if 0 (`UNSPECIFIED`) is used, methods that perform spatial operations involving another input SRF will produce an error condition (`OPERATION_UNSUPPORTED`), if the method does not also require an $\boldsymbol{H}_{\mathbf{ST}}$ input.

Inputs	`srfs_code_info:` `SRFS_Code_Info`
	`orm_code:` `ORM_Code`
	`rt_code:` `RT_Code`

Outputs	`new_srf:` (concrete subclass of) `BaseSRF`
Error conditions	1) `UNDEFINED_CODE` if the value of `orm_code`, `rt_code`, the SRF set code in `srfs_code_info`, or the SRF set member code in `srfs_code_info` is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) `INVALID_CODE` if (1) the `orm_code` has the `ORM_Code` value 0 (`UNSPECIFIED`), (2) the SRF set code in `srfs_code_info` has the `SRFS_Code` value 0 (`UNSPECIFIED`), or (3) the SRF set member code in `srfs_code_info` has the SRF set member code type value 0 (`UNSPECIFIED`). 3) `INCOMPATIBLE_CODE` if (1) the value of the SRF set member code in `srfs_code_info` is not compatible with the SRF set in `srfs_code_info`, (2) the ORM specified by `orm_code` is not compatible with the ORM constraints of the SRF set specified in `srfs_code_info`, or (3) the value of `rt_code` is not compatible with the value of `orm_code`..

11.4.3 Implementation support query functions

Several query functions are provided to indicate the support of an API implementation for subsets of the SRM as may be defined by a profile (see Clause 12 and 14.2). The functions `QueryProfileSupportList` and `QueryProfileSupport` indicate which SRM profiles are supported. The functions `QueryORMSupportList`, `QueryORMSupport` and `QueryRTSupportList` indicate which ORMs, RTs and ORM-RT combinations are supported. The functions `QuerySRFTSupportList` and `QuerySRFTSupport` indicate which SRF template classes are supported. The functions `QuerySRFSupportList` and `QuerySRFSupport` indicate which SRF classes are supported. The functions `QuerySRFSetSupportList` and `QuerySRFSetSupport` indicate which SRF set classes are supported. The functions `QueryDSSSupportList` and `QueryDSSSupport` indicate which DSS classes are supported.

Table 11.52 — QueryDSSSupport

Element	Specification
Function	`QueryDSSSupport`
Semantics	If the implementation supports the full functionality and all the associated data types of the DSS indicated by the input `dss_code`, then the output parameter `supported` is set to the `Boolean` value `true`. Otherwise, `supported` is set to the `Boolean` value `false`.
Inputs	`dss_code:` `DSS_Code`

Element	Specification
Function	QueryDSSSupport
Outputs	supported: Boolean
Error conditions	1) UNDEFINED_CODE if the value of dss_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if dss_code has the DSS_Code value 0 (UNSPECIFIED).

Table 11.53 — QueryDSSSupportList

Element	Specification
Function	QueryDSSSupportList
Semantics	Returns lists of DSS codes identifying the DSSs that are supported by the implementation.
Inputs	none
Outputs	supported_dss_codes: DSS_Code_Array
Error conditions	No additional error conditions.

Table 11.54 — QueryGeodeticRTRegionSpecification

Element	Specification
Function	QueryGeodeticRTRegionSpecification
Semantics	Returns the geodetic RT region specification, if any, for the specified ORM and RT. If the RT region is specified in this International Standard, is_set is returned as true, and the latitude and longitude coordinate-component intervals are returned. If the RT region is not specified, is_set is returned as false, and the latitude and longitude coordinate-component intervals are returned as UNBOUNDED.
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	is_set: Boolean latitude_interval: Interval longitude_interval: Interval
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if the value of rt_code is not compatible with the value of orm_code.

Table 11.55 — QueryORMSupport

Element	Specification
Function	QueryORMSupport
Semantics	If the implementation supports the parameter values and all the associated data types of the ORM and the RT indicated by the inputs orm_code, and rt_code, then the output supported is set to the Boolean value true. Otherwise, the output supported is set to the Boolean value false. The rt_code value 0 (UNSPECIFIED) is permitted. In that case, the output supported is set to the Boolean value true only if the implementation supports the parameter values and all the associated data types of the ORM indicated by the input orm_code.

Element	Specification
Function	QueryORMSupport
Inputs	orm_code: ORM_Code rt_code: RT_Code
Outputs	supported: Boolean
Error conditions	1) UNDEFINED_CODE if the value of orm_code or rt_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED). 3) INCOMPATIBLE_CODE if the value of rt_code is not compatible with the value of orm_code.

Table 11.56 — QueryORMSupportList

Element	Specification
Function	QueryORMSupportList
Semantics	Returns a list of ORM codes identifying the ORMs that are supported by the implementation.
Inputs	none
Outputs	supported_orm_codes: ORM_Code_Array
Error conditions	No additional error conditions.

Table 11.57 — QueryProfileSupport

Element	Specification
Function	QueryProfileSupport
Semantics	If the implementation supports the full functionality and all the associated data types of the profile indicated by the input profile_code, then the output parameter supported is set to the Boolean value true. Otherwise, supported is set to the Boolean value false.
Inputs	profile_code: Profile_Code
Outputs	supported: Boolean
Error conditions	UNDEFINED_CODE if the value of profile_code is (1) not defined by this International Standard, or (2) not defined by this implementation.

Table 11.58 — QueryProfileSupportList

Element	Specification
Function	QueryProfileSupportList
Semantics	Returns a list, possibly empty, of profile codes identifying the profiles that are supported by the implementation.
Inputs	none
Outputs	supported_profile_codes: Profile_Code_Array
Error conditions	No additional error conditions.

Table 11.59 — QueryRTSupportList

Element	Specification
Function	QueryRTSupportList
Semantics	Returns a list of RT codes identifying the RTs that are associated with the specified ORM and are supported by the implementation.
Inputs	orm_code: ORM_Code
Outputs	supported_rt_codes: RT_Code_Array
Error conditions	1) UNDEFINED_CODE if the value of orm_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if orm_code has the ORM_Code value 0 (UNSPECIFIED).

Table 11.60 — QuerySRFSetSupport

Element	Specification
Function	QuerySRFSetSupport
Semantics	If the implementation supports the full functionality and all the associated data types of the SRF set indicated by the input srfs_code, then the output parameter supported is set to the Boolean value true. Otherwise, supported is set to the Boolean value false.
Inputs	srfs_code: SRFS_Code
Outputs	supported: Boolean
Error conditions	1) UNDEFINED_CODE if the value of srfs_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if srfs_code has the SRFS_Code value 0 (UNSPECIFIED).

Table 11.61 — QuerySRFSetSupportList

Element	Specification
Function	QuerySRFSetSupportList
Semantics	Returns lists of SRF set codes identifying the SRF sets that are supported by the implementation.
Inputs	none
Outputs	supported_srfs_codes: SRFS_Code_Array
Error conditions	No additional error conditions.

Table 11.62 — QuerySRFSupport

Element	Specification
Function	QuerySRFSupport
Semantics	If the implementation supports the full functionality and all the associated data types of the SRF class indicated by the input srf_code, then the output parameter supported is set to the Boolean value true. Otherwise, supported is set to the Boolean value false.
Inputs	srf_code: SRF_Code
Outputs	supported: Boolean
Error conditions	1) UNDEFINED_CODE if the value of srf_code is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) INVALID_CODE if srf_code has the SRF_Code value 0 (UNSPECIFIED).

Table 11.63 — QuerySRFSupportList

Element	Specification
Function	`QuerySRFSupportList`
Semantics	Returns lists of SRF codes identifying the SRFs that are supported by the implementation.
Inputs	none
Outputs	`supported_srf_codes:` `SRF_Code_Array`
Error conditions	No additional error conditions.

Table 11.64 — QuerySRFTSupport

Element	Specification
Function	`QuerySRFTSupport`
Semantics	If the implementation supports the full functionality and all the associated data types of the SRF class indicated by the input `srft_code`, then the output parameter `supported` is set to the `Boolean` value `true`. Otherwise, `supported` is set to the `Boolean` value `false`.
Inputs	`srft_code:` `SRFT_Code`
Outputs	`supported:` `Boolean`
Error conditions	1) `UNDEFINED_CODE` if the value of `srft_code` is (1) not defined by this International Standard, or (2) not defined by this implementation. 2) `INVALID_CODE` if `srft_code` has the `SRFT_Code` value 0 (`UNSPECIFIED`).

Table 11.65 — QuerySRFTSupportList

Element	Specification
Function	`QuerySRFTSupportList`
Semantics	Returns lists of SRFT codes identifying the SRFTs that are supported by the implementation.
Inputs	none
Outputs	`supported_srft_codes:` `SRFT_Code_Array`
Error conditions	No additional error conditions.

Table 11.66 — QueryVersion

Element	Specification
Function	`QueryVersion`
Semantics	Returns the version of this International Standard implemented by this API and the version of the implementation. Both of the versions are returned as strings. The version of the International Standard is in the form "e_a", where "e" indicates the Edition number and "a" indicates the highest-numbered amendment used by the implementation. If no amendment is used by the implementation, the "a" is set to zero. The form of the implementation version is implementation-dependent.
Inputs	none
Outputs	`standard_version:` `String` `implementation_version:` `String`
Error conditions	No additional error conditions.

See also common error conditions in 11.2.7.12.

11.5 Data storage structures

11.5.1 Introduction

The data storage structures specify the exact ordering sequence and size of the information for persistent storage in any mass storage media. These structures are defined within this International Standard for applications that store SRM data and are not used with API methods or functions defined in this International Standard.

11.5.2 Selection data types

11.5.2.1 OBRS_Code

This selection data type specifies an OBRS code as defined in Clause 7. Table 7.36 is a directory of OBRS specifications, each of which includes a code value and a corresponding label. An OBRS is an SRM concept that is not directly used in the functional API.

11.5.2.2 RD_Code

This selection data type specifies the RD code associated with a specified RD as defined in Clause 7. Table 7.3 is a directory of RD specifications, each of which includes a code value and a corresponding label. A standardized or registered RD is represented by its RD code.

11.5.2.3 Spatial_Coordinate_Code

This selection data type specifies different types of spatial coordinates.

Values less than zero are reserved for use by implementations. Values greater than 37 are reserved for registration.

```
Spatial_Coordinate_Code ::= (    0  : UNSPECIFIED,
                                 1  : CC_3D,
                                 2  : CD_3D,
                                 3  : CD_SURFACE,
                                 4  : CM_3D,
                                 5  : EC_AUGMENTED_3D,
                                 6  : EC_SURFACE,
                                 7  : EI_3D,
                                 8  : HAEC_3D,
                                 9  : HEEC_3D,
                                 10 : HEEQ_3D,
                                 11 : LCC_AUGMENTED_3D,
                                 12 : LCC_SURFACE,
                                 13 : LCE_3D,
                                 14 : LSA_2D,
                                 15 : LSP_2D,
                                 16 : LSR_2D,
                                 17 : LSR_3D,
                                 18 : LTSAS_3D,
                                 19 : LTSAS_SURFACE,
                                 20 : LTSC_3D,
                                 21 : LTSC_SURFACE,
```

```
                                22 : LTSE_3D,
                                23 : LTSE_SURFACE,
                                24 : M_AUGMENTED_3D,
                                25 : M_SURFACE,
                                26 : OMS_AUGMENTED_3D,
                                27 : OMS_SURFACE,
                                28 : PD_3D,
                                29 : PD_SURFACE,
                                30 : PS_AUGMENTED_3D,
                                31 : PS_SURFACE,
                                32 : SEC_3D,
                                33 : SEQ_3D,
                                34 : SMD_3D,
                                35 : SME_3D,
                                36 : TM_AUGMENTED_3D,
                                37 : TM_SURFACE )
```

11.5.2.4 SRF_Parameters_Info_Code

This selection data type specifies different ways of identifying an arbitrary SRF; as an instance of a template, as an SRF set or as a standardized SRF.

Values less than zero are reserved for use by implementations. Values greater than 3 are reserved for registration.

```
SRF_Parameters_Info_Code ::= ( 1 : TEMPLATE,
                               2 : SET,
                               3 : INSTANCE )
```

11.5.3 Record data types

11.5.3.1 Coordinate structures

Structures are defined to store the specific values of a coordinate.

11.5.3.1.1 CC_3D_Coordinate

This record data type specifies the 3D coordinate-components for SRFT CELESTIOCENTRIC.

```
CC_3D_Coordinate ::= {
   x                                Long_Float;
   y                                Long_Float;
   z                                Long_Float;
}
```

11.5.3.1.2 CD_3D_Coordinate

This record data type specifies the 3D coordinate-components for SRFT CELESTIODETIC.

```
CD_3D_Coordinate ::= {
   longitude                        Long_Float;
   latitude                         Long_Float;
   ellipsoidal_height               Long_Float;
}
```

11.5.3.1.3 CD_Surface_Coordinate

This record data type specifies the surface coordinate-components for SRFT CELESTIODETIC.

```
CD_Surface_Coordinate ::= {
   longitude                        Long_Float;
   latitude                         Long_Float;
}
```

11.5.3.1.4 EI_3D_Coordinate

This record data type specifies the 3D coordinate-components for SRFT EQUATORIAL_INERTIAL.

```
EI_3D_Coordinate ::= {
   right_ascension                  Long_Float;
   declination                      Long_Float;
   radius                           Long_Float;
}
```

11.5.3.1.5 Equatorial_Spherical_3D_Coordinate

This record data type specifies the 3D coordinate-components for equatorial spherical SRFTs.

```
Equatorial_Spherical_3D_Coordinate ::= {
   longitude                        Long_Float;
   latitude                         Long_Float;
   radius                           Long_Float;
}
```

11.5.3.1.6 Euclidean_2D_Coordinate

This record data type specifies the 2D coordinate-components for Euclidean space SRFTs.

```
Euclidean_2D_Coordinate ::= {
   u                                Long_Float;
   v                                Long_Float;
}
```

11.5.3.1.7 Euclidean_3D_Coordinate

This record data type specifies the 3D coordinate-components for Euclidean space SRFTs.

```
Euclidean_3D_Coordinate ::= {
   u                                Long_Float;
   v                                Long_Float;
   w                                Long_Float;
}
```

11.5.3.1.8 LSA_2D_Coordinate

This record data type specifies the 2D coordinate-components for SRFT LOCAL_SPACE_AZIMUTHAL_2D.

```
LSA_2D_Coordinate ::= {
   azimuth                          Long_Float;
   radius                           Long_Float;
}
```

11.5.3.1.9 LSP_2D_Coordinate

This record data type specifies the surface coordinate-components for SRFT LOCAL_SPACE_POLAR_2D.

```
LSP_2D_Coordinate ::= {
   radius                          Long_Float;
   angle                           Long_Float;
}
```

11.5.3.1.10 LTSAS_3D_Coordinate

This record data type specifies the 3D coordinate-components for SRFT LOCAL_TANGENT_SPACE_AZIMUTHAL_SPHERICAL.

```
LTSAS_3D_Coordinate ::= {
   azimuth                          Long_Float;
   radius                           Long_Float;
   angle                            Long_Float;
}
```

11.5.3.1.11 LTSAS_Surface_Coordinate

This record data type specifies the surface coordinate-components for SRFT LOCAL_TANGENT_SPACE_AZIMUTHAL_SPHERICAL.

```
LTSAS_Surface_Coordinate ::= {
   azimuth                          Long_Float;
   radius                           Long_Float;
}
```

11.5.3.1.12 LTSC_3D_Coordinate

This record data type specifies the 3D coordinate-components for SRFT LOCAL_TANGENT_SPACE_CYLINDRICAL.

```
LTSC_3D_Coordinate ::= {
   radius                          Long_Float;
   angle                           Long_Float;
   height                          Long_Float;
}
```

11.5.3.1.13 LTSC_Surface_Coordinate

This record data type specifies the surface coordinate-components for SRFT LOCAL_TANGENT_SPACE_CYLINDRICAL.

```
LTSC_Surface_Coordinate ::= {
   radius                          Long_Float;
   angle                           Long_Float;
}
```

11.5.3.1.14 LTSE_3D_Coordinate

This record data type specifies the 3D coordinate-components for SRFT LOCAL_TANGENT_SPACE_EUCLIDEAN.

```
LTSE_3D_Coordinate ::= {
   x                                Long_Float;
   y                                Long_Float;
   height                           Long_Float;
}
```

11.5.3.1.15 LTSE_Surface_Coordinate

This record data type specifies the surface coordinate-components for SRFT LOCAL_TANGENT_SPACE_EUCLIDEAN.

```
LTSE_Surface_Coordinate ::= {
   x                                Long_Float;
   y                                Long_Float;
}
```

11.5.3.1.16 Map_Projection_3D_Coordinate

This record data type specifies the 3D coordinate-components for map projection SRFTs.

```
Map_Projection_3D_Coordinate ::= {
   easting                          Long_Float;
   northing                         Long_Float;
   llipsoidal_height                Long_Float;
}
```

11.5.3.1.17 Map_Projection_Surface_Coordinate

This record data type specifies the surface coordinate-components for map projection SRFTs.

```
Map_Projection_Surface_Coordinate ::= {
   easting                          Long_Float;
   northing                         Long_Float;
}
```

11.5.3.1.18 PD_3D_Coordinate

This record data type specifies the 3D coordinate-components for SRFT PLANETODETIC.

```
PD_3D_Coordinate ::= {
   latitude                         Long_Float;
   pd_longitude                     Long_Float;
   ellipsoidal_height               Long_Float;
}
```

11.5.3.1.19 PD_Surface_Coordinate

This record data type specifies the surface coordinate-components for SRFT PLANETODETIC.

```
PD_Surface_Coordinate ::= {
   latitude                         Long_Float;
   pd_longitude                     Long_Float;
}
```

11.5.3.2 Coordinate

This variant record data type stores one of the defined coordinates.

```
Coordinate ::=  (   spatial_coord_code  Spatial_Coordinate_Code )
{
 [
   CC_3D:                  cc_3d           CC_3D_Coordinate;
   CD_3D:                  cd_3d           CD_3D_Coordinate;
   CD_SURFACE:             cd_surface      CD_Surface_Coordinate;
   CM_3D:                  cm_3d           Equatorial_Spherical_3D_Coordinate;
   EC_AUGMENTED_3D:        ec_aug_3d       Map_Projection_3D_Coordinate;
   EC_SURFACE:             ec_surface      Map_Projection_Surface_Coordinate;
   EI_3D:                  ei_3d           Equatorial_Inertial_3D_Coordinate;
   HAEC_3D:                haec_3d         Equatorial_Spherical_3D_Coordinate;
   HEEC_3D:                heec_3d         Equatorial_Spherical_3D_Coordinate;
   HEEQ_3D                 heeq_3d         Equatorial_Spherical_3D_Coordinate;
   LCC_AUGMENTED_3D        lcc_aug_3d      Map_Projection_3D_Coordinate;
   LCC_SURFACE:            lcc_surface     Map_Projection_Surface_Coordinate;
   LCE_3D:                 lce_3d          Euclidean_3D_Coordinate;
   LSA_2D:                 lsa_2d          LSA_2D_Coordinate;
   LSP_2D:                 lsp_2d          LSP_2D_Coordinate;
   LSR_2D:                 lsr_2d          Euclidean_2D_Coordinate;
   LSR_3D:                 lsr_3d          Euclidean_3D_Coordinate;
   LTSAS_3D:               ltsas_3d        LTSAS_3D_Coordinate;
   LTSAS_SURFACE:          ltsas_surface   LTSAS_Surface_Coordinate;
   LTSC_3D:                ltsc_3d         LTSC_3D_Coordinate;
   LTSC_SURFACE:           ltsc_surface    LTSC_Surface_Coordinate;
   LTSE_3D:                ltse_3d         LTSE_3D_Coordinate;
   LTSE_SURFACE:           ltse_surface    LTSE_Surface_Coordinate;
   MERCATOR_AUGMENTED_3D:  m_aug_3d        Map_Projection_3D_Coordinate;
   MERCATOR_SURFACE:       m_surface       Map_Projection_Surface_Coordinate;
   OMS_AUGMENTED_3D:       oms_aug_3d      Map_Projection_3D_Coordinate;
   OMS_SURFACE:            oms_surface     Map_Projection_Surface_Coordinate;
   PD_3D:                  pd_3d           PD_3D_Coordinate;
   PD_SURFACE:             pd_surface      PD_Surface_Coordinate;
   PS_AUGMENTED_3D:        ps_aug_3d       Map_Projection_3D_Coordinate;
   PS_SURFACE:             ps_surface      Map_Projection_Surface_Coordinate;
   SEC_3D:                 sec_3d          Equatorial_Spherical_3D_Coordinate;
   SEQ_3D:                 seq_3d          Equatorial_Spherical_3D_Coordinate;
   SMD_3D:                 smd_3d          Euclidean_3D_Coordinate;
   SME_3D:                 sme_3d          Euclidean_3D_Coordinate;
   TM_AUGMENTED_3D:        tm_aug_3d       Map_Projection_3D_Coordinate;
   TM_SURFACE:             tm_surface      Map_Projection_Surface_Coordinate;
 ]
}
```

11.5.3.3 SRF_Parameters_Info

This variant record data type specifies the parameters for an arbitrary SRF, SRF set or SRF template.

```
SRF_Parameters_Info ::= ( srf_parameters_info_code    SRF_Parameters_Info_Code )
{
  rt_code           RT_Code;
  [TEMPLATE:        srf_template          SRFT_Parameters;
   SET:             srf_set               SRFS_Info;
   INSTANCE:        srf_instance          SRF_Code;
```

```
 ]
}
```

11.5.3.4 SRF_Reference_Surface_Info

This record data type specifies the information for an arbitrary SRF with its associated DSS information.

```
SRF_Reference_Surface_Info ::= {
   dss_code                          DSS_Code;
   srf_parameters_info               SRF_Parameters_Info;
}
```

11.5.3.5 SRFS_Info

This record data type specifies the parameters for an arbitrary SRF set.

```
SRFS_Info ::= {
   orm_code                          ORM_Code;
   srfs_code_info                    SRFS_Code_Info;
}
```

11.5.3.6 SRFT_Parameters

This variant record data type specifies the parameters for an arbitrary SRF template.

```
SRFT_Parameters ::= ( template_code     SRFT_Code )
{
  orm_code         ORM_Code;
    [
    CELESTIOCENTRIC:
      cc_srf_parameters               <empty>;
    LOCAL_SPACE_RECTANGULAR_3D:
      lsr_3d_srf_parameters           LSR_3D_Parameters;
    CELESTIODETIC:
       cd_srf_parameters              <empty>;
    PLANETODETIC:
      pd_srf_parameters               <empty>;
    LOCAL_TANGENT_SPACE_EUCLIDEAN:
      ltse_srf_parameters             LTSE_Parameters;
    LOCAL_TANGENT_SPACE_AZIMUTHAL_SPHERICAL:
      ltsas_srf_parameters            Local_Tangent_Parameters;
    LOCAL_TANGENT_SPACE_CYLINDRICAL:
      ltsc_srf_parameters             Local_Tangent_Parameters;
    LOCOCENTRIC_EUCLIDEAN_3D:
      lce_3d_srf_parameters           LCE_3D_Parameters;
    CELESTIOMAGNETIC:
      cm_srf_parameters               <empty>;
    EQUATORIAL_INERTIAL:
      ei_srf_parameters               <empty>;
    SOLAR_ECLIPTIC:
      sec_srf_parameters              <empty>;
    SOLAR_EQUATORIAL:
      seq_srf_parameters              <empty>;
    SOLAR_MAGNETIC_ECLIPTIC:
      sme_srf_parameters              <empty>;
```

```
    SOLAR_MAGNETIC_DIPOLE:
      smd_srf_parameters              <empty>;
    HELIOSPHERIC_ARIES_ECLIPTIC:
      haec_srf_parameters             <empty>;
    HELIOSPHERIC_EARTH_ECLIPTIC:
      heec_srf_parameters             <empty>;
    HELIOSPHERIC_EARTH_EQUATORIAL:
      heeq_srf_parameters             <empty>;
    MERCATOR:
      m_srf_parameters                M_Parameters;
    OBLIQUE_MERCATOR_SPHERICAL:
      oms_srf_parameters              Oblique_Mercator_Parameters;
    TRANSVERSE_MERCATOR:
      tm_srf_parameters               TM_Parameters;
    LAMBERT_CONFORMAL_CONIC:
      lcc_srf_parameters              LCC_Parameters;
    POLAR_STEREOGRAPHIC:
      ps_srf_parameters               PS_Parameters;
    EQUIDISTANT_CYLINDRICAL:
      ec_srf_parameters               EC_Parameters;
    LOCAL_SPACE_RECTANGULAR_2D:
      lsr_2d_srf_parameters           LSR_2D_Parameters;
    LOCAL_SPACE_AZIMUTHAL:
      lsa_srf_parameters              <empty>;
    LOCAL_SPACE_POLAR:
      lsp_srf_parameters              <empty>;
  ]
}
```