11 Application program interface

11.1 Introduction

This International Standard specifies an API for the SRF operations in Clause 5 and Clause 10. The API specifies non-object data types (see 11.2), and object classes (see 11.3) used to perform the spatial operations. Two functions are provided to create certain object instances (see 11.4 and 11.5). Two query functions are also provided to indicate the extent of support of an API implementation for a profile of the SRM (see 11.6 and Clause 12). The API also specifies data storage structures for the representation of SRM concepts that are not used to perform spatial operations (see 11.9).

Class is the term used to categorize the general form of *object* instances. Each class definition specifies the *methods* (if any) that operate on the object. Methods are specified by giving their syntax (input and output parameters), semantics (how the inputs interact with the *state* of an instance of the class and produce any *outputs*), and error conditions. In particular, the state of an instance of the class is implicitly an input for each of its methods with the exception of the `Create` method. The `Create` method of an object depends only on its explicit inputs. The state of a class instance may change only as the result of applying a method of the class.

The active objects created as instances of a given class are reliably denoted by *object references*. Once created, objects exist and respond to method invocations until they are destroyed. The property of being created and existing until destruction is termed the *object life cycle*. Classes inherit methods from other classes through the *subclass*/*superclass* relationship. Method inheritance is transitive: a subclass also inherits the methods that have been inherited by its superclass.

Non-object data types do not have an object life cycle nor do they have operations other than those defined by a programming language that this API might be bound to.

EXAMPLE `Integer` is a non-object data type. Programming languages to which this API may be bound have definition mechanisms and operations for creating and then performing arithmetic operations on integers as variables and/or constants in the programming language.

The API specifies seven abstract classes (see 11.3.3 and 11.3.5):

a) `LifeCycleObject`,

b) `BaseSRF`,

c) `BaseSRF2D`,

d) `BaseSRF3D`,

e) `BaseSRFwithTangentPlaneSurface`,

f) `BaseSRFwithEllipsoidalHeight`, and

g) `BaseSRFMapProjection`.

These abstract classes are used as base classes from which subclasses including concrete classes inherit common sets of methods. `LifeCycleObject` includes the creation and destruction methods that all other classes inherit. The `LifeCycleObject` creation method specification may be overridden in a concrete subclass to provide subclass specific inputs and error conditions. The use of abstract classes in this International Standard is solely for the purpose of specifying common methods in only one place instead of repeating the same specification in each concrete class for which it applies. API implementations are not required to implement abstract classes.

The API specifies six classes whose methods are not exposed as part of the API:

a) three coordinate classes: Coordinate2D, Coordinate3D, and SurfaceCoordinate;

b) one direction class Direction and

c) two position classes: Position2D, and Position3D.

These classes are private classes that hide all aspects of the implementation of instances of these objects from the application.

The API specifies a set of concrete classes that correspond to specific SRFTs specified in Clause 8 (see 11.3.6 through 11.3.10). An instance of one of these concrete classes corresponds to a specific SRF.

Instances of concrete SRF classes that correspond to the coded collection of SRFT instances specified in Table 8.30 are created by the function CreateStandardSRF. CreateStandardSRF takes the corresponding SRF_Code (see 11.4) as an input.

Instances of concrete SRF classes that correspond to the members of SRF Sets specified in Table 8.31 are also created by the function CreateSRFSetMember. CreateSRFSetMember takes an SRFS_Code_Info (see 11.5) as an input.

The class hierarchy is illustrated in Figure 11.1. Procedural rules for using LifeCycleObjects in applications and examples of use of the API are provided in 11.8.

11.2 Non-object data types

11.2.1 Overview

Basic non-object data types represent single pieces of information such as numbers, codes, and other individual data items. Structured data types represent data records of basic non-object 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 non-object types.

Table 11.1 — SRFT abbreviations

Abbreviation	SRFT
CC	Celestiocentric
CD	Celestiodetic
CM	Celestiomagnetic
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

Abbreviation	SRFT
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_Positive` and `Integer`. 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_Positive`	[1, 4 294 967 295]
`Integer`	[-2 147 483 647, 2 147 483 647]

`Long_Float` is a non-object data type defined for floating-point numbers. This data type corresponds to the double precision floating-point data type specified by IEC 60559. However, implementations on architectures that support other floating-point representations are allowed.

11.2.4 Logicals

The general-purpose logical data type is Boolean. All implementations that conform to this standard shall support this 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 an opaque non-object data type that allows an application to reliably access an instance of an object. `Object_Reference`s may be compared for equality and tested to see if they are equal to the special value `NULL_Object`. If two `Object_Reference`s are equal, they refer to the same object instance. If an `Object_Reference` is equal to the special value `NULL_Object` it does not reference any object instance. In all the method specifications in this clause, whenever an argument passed to or returned from a method is an object, it is an object reference that is passed.

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. The names are assigned numbers whose values indicate the position within the ordered list. It is these numbers that are actually manipulated by the implementation. Enumerated data types are a closed list the members of which do not change based on registration or deprecation. This clause specifies the enumerated data types within this International Standard.

11.2.6.2 Axis_Direction

This data type represents the values of the axis direction parameter(s) of the SRFTs LOCAL_SPACE_RECTANGULAR_3D and LOCAL_SPACE_RECTANGULAR_2D.

```
Axis_Direction ::= (  POSITIVE_PRIMARY_AXIS,
                      POSITIVE_SECONDARY_AXIS,
                      POSITIVE_TERTIARY_AXIS,
                      NEGATIVE_PRIMARY_AXIS,
                      NEGATIVE_SECONDARY_AXIS,
                      NEGATIVE_TERTIARY_AXIS )
```

11.2.6.3 Coordinate_Valid_Region

This data type represents coordinate location with respect to valid-regions (see 8.3.2.4).

```
Coordinate_Valid_Region ::= ( VALID,
                              EXTENDED_VALID,
                              DEFINED )
```

`VALID` denotes a coordinate that is contained in the valid-region and in the CS domain.
`EXTENDED_VALID` denotes a coordinate that is contained in the extended valid-region and in the CS domain but not in the valid-region.
`DEFINED` denotes a coordinate that is contained in the CS domain but not in the valid or the extended valid-regions.

11.2.6.4 Interval_Type

This data type is used to specify coordinate-component intervals in the `SetValidRegion`, `SetExtendedValidRegion`, `GetValidRegion`, and `GetExtendedValidRegion` methods of class `BaseSRF3D` and in the `SetValidGeodeticRegion`, `SetExtendedValidGeodeticRegion`, `GetValidGeodeticRegion`, and `GetExtendedValidGeodeticRegion` methods of class `BaseSRFMapProjection`.

```
Interval_Type::= ( OPEN_INTERVAL,      // The bounded open interval (a, b).
                   GE_LT_INTERVAL,     // The bounded interval [a, b).
                   GT_LE_INTERVAL,     // The bounded interval (a, b].
                   CLOSED_INTERVAL,    // The bounded interval [a, b].
                   GT_SEMI_INTERVAL,   // The unbounded interval (a, +infinity).
                   GE_SEMI_INTERVAL,   // The unbounded interval [a, +infinity).
                   LT_SEMI_INTERVAL,   // The unbounded interval (-infinity, b).
                   LE_SEMI_INTERVAL,   // The unbounded interval (-infinity, b].
                   UNBOUNDED           // All values (-infinity, +infinity)
                 )
```

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 )
```

11.2.7 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. Selection data types are all defined to be as distinct sub-data types of the numeric of data type `Integer`, but with specific meanings attached to each value. The set of selections may be augmented by assigning meanings to additional values. Selection data types are otherwise processed in the same manner as enumerated data types. The integer codes are unique within each concept set, but not between sets. Although the `RT_Code` is used in combination with an `ORM_Code`, its code space follows the general rule and is independent of the `ORM_Code`.

In each code space the valid `Integer` values are 0 and greater. Negative code values are implementation dependent and non-conforming. In each code space, the `Integer` value 0 (`UNSPECIFIED`) is reserved. Some API methods and functions allow 0 (`UNSPECIFIED`) as an input `Integer` code value and/or an output `Integer` 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 `Integer` code 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 `Integer` code 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 `Integer` code 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 Integer code data type ORMT_Code specifies an ORM Template code defined in Clause 7 or by registration. Table 7.12 is a directory of ORMT specifications, each of which includes a code value and a corresponding label.

11.2.7.6 RT_Code

The Integer code data type RT_Code specifies a reference transformation $\boldsymbol{H}_{\mathbf{SR}}$. Each RT_Code is defined 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.

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

11.2.7.7 SRF_Code

The Integer code data type SRF_Code specifies an SRF by its code as defined in Table 8.30 or by registration. Each SRF specification includes a code value and a corresponding label (see Clause 8).

11.2.7.8 SRFS_Code

The Integer code 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).

```
SRFS_Code ::= (        < 0 :   // implementation_dependent,
                         0 :   SRFS_UNSPECIFIED,
                         1 :   SRFS_ALABAMA_SPCS,
                         2 :   SRFS_GTRS_GLOBAL_COORDINATE_SYSTEM,
                         3 :   SRFS_JAPAN_RECTANGULAR_PLANE_CS,
                         4 :   SRFS_LAMBERT_NTF,
                         5 :   SRFS_UNIVERSAL_POLAR_STEREOGRAPHIC,
                         6 :   SRFS_UNIVERSAL_TRANSVERSE_MERCATOR,
                         7 :   SRFS_WISCONSIN_SPCS,
                        >7 :   // reserved for registration )
```

11.2.7.9 SRFS member types

11.2.7.9.1 Introduction

The Integer code types that specify the SRFS members associated with the SRFS defined in Table 8.48.

11.2.7.9.2 SRFSM_Alabama_SPCS_Code

The Integer code data type SRFSM_Alabama_SPCS_Code specifies a member of the Alabama SPCS SRFS in Table 8.50 or by registration.

11.2.7.9.3 SRFSM_GTRS_Global_Coordinate_System _Code

The Integer code data type SRFSM_GTRS_Global_Coordinate_System_Code specifies a member of the GTRS Global Coordinate System SRFS in Table 8.52 and Table 8.53 or by registration.

11.2.7.9.4 SRFSM_Japan_Rectangular_Plane_CS_Code

The Integer code data type SRFSM_Japan_Rectangular_Plane_CS_Code specifies a member of the Japan Rectangular Plane CS SRFS in Table 8.55 or by registration.

11.2.7.9.5 SRFSM_Lambert_NTF_Code

The Integer code data type SRFSM_Lambert_NTF_Code specifies a member of the Lambert NTF SRFS in Table 8.57 or by registration.

11.2.7.9.6 SRFSM_Universal_Polar_Stereographic_Code

The Integer code data type SRFSM_Universal_Polar_Stereographic_Code specifies a member of the Universal Polar Stereographic SRFS in Table 8.59 or by registration.

11.2.7.9.7 SRFSM_Universal_Transverse_Mercator_Code

The Integer code data type SRFSM_Universal_Transverse_Mercator_Code specifies a member of the Universal Transverse Mercator SRFS in Table 8.61 or by registration.

11.2.7.9.8 SRFSM_Wisconsin_SPCS _Code

The Integer code data type SRFSM_Wisconsin_SPCS_Code specifies a member of the Wisconsin SPCS SRFS Table 8.63 or by registration.

11.2.7.10 SRFT_Code

The Integer code 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.

11.2.7.11 Status_Code

The Status_Code non-object selection data type specifies the status codes associated with methods on instances of classes specified in this International Standard. The meaning of values other than SUCCESS varies according to the class and method or function and is further defined in the “Error conditions” element of each method or function specification in Table 11.6 through Table 11.48 (see common error conditions in 11.3.2). This selection data type may be extended in a language binding specification.

```
Status_Code ::= (   < 0 :   // implementation_dependent,
                      0 :   UNSPECIFIED,    // reserved
                      1 :   SUCCESS,    // the operation was performed successfully
                      2 :   INVALID_SRF,
                      3 :   INVALID_SOURCE_SRF,
                      4 :   INVALID_SOURCE_COORDINATE,
                      5 :   INVALID_TARGET_COORDINATE,
                      6 :   INVALID_POINT1_COORDINATE,
                      7 :   INVALID_POINT2_COORDINATE,
                      8 :   OPERATION_UNSUPPORTED,
                      9 :   INVALID_SOURCE_DIRECTION,
                     10 :   INVALID_TARGET_DIRECTION,
                     11 :   INVALID_CODE,
                     12 :   INVALID_INPUT,
                     13 :   CREATION_FAILURE,
 14 :   DESTRUCTION_FAILURE,
 15 :   FLOATING_OVERFLOW,
 16 :   FLOATING_UNDERFLOW,
 17 :   FLOATING_POINT_ERROR,
 18 :   MEMORY_ALLOCATION_ERROR,
>18 :   // reserved for language binding specification )
```

11.2.8 Array data types

11.2.8.1 Introduction

Array data types specify an ordered set whose elements may be of any single 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, cloumns]

The symbols “length”, "rows", and "columns" are positive integers. The length of a one-dimensional array is specified by "length". When the length is specified by another field of a record data type or by a function parameter, the field name or function parameter name that will be used to indicate that the size of the array is obtained from the value of that construct. The index of the first element in the 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 array are both either “0” or “1” depending on the language binding.

11.2.8.2 Coordinate2D_Array

This data type specifies an array of `Coordinate2D` objects.

```
Coordinate2D_Array ::= {

   length                         Integer_Positive;
   coordinate2D_array             Object_Reference[ length ];
}
```

11.2.8.3 Coordinate3D_Array

This data type specifies an array of `Coordinate3D` objects.

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

11.2.8.4 Coordinate_Valid_Region_Array

This data type specifies an array of `Coordinate_Valid_Region` variables.

```
Coordinate_Valid_Region_Array ::= {
   length                         Integer_Positive;
   valid_region_array              Coordinate_Valid_Region[ length ];
}
```

11.2.8.5 Direction_Array

This data type specifies an array of `Direction` objects.

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

11.2.8.6 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 ]
```

11.2.8.7 Matrix_3x3

This data type specifies a two-dimensional square array of nine `Long_Float` variables representing a 3x3 matrix (see 10.4.6).

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

11.2.8.8 Matrix_4x4

This data type specifies a two-dimensional square array of 16 `Long_Float` variables representing a 4x4 matrix (see 10.4.6).

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

11.2.9 Structured data types

11.2.9.1 Introduction

Non-object data types created as records whose elements are basic non-object data types are called *structured non-object data types*. This International Standard specifies a set of structured non-object data types to collect the (non-ORM) parameters needed to specify an SRF by means of an SRF template, and to collect parameters needed to specify an ORM transformation.

The elements of structured data types that represent lengths shall be evaluated in the units of metre, and the elements that represent angles shall be evaluated in the units of radian.

The following notation is used for defining the variant record data structures for non-object 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:

<Variant_Record_Data_Type>: The variant record data type that is being defined.
<Selector_Name>: The name of the selector
<Selector_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>" signifies the element is not present in the record.
<Selection_Name>: A selection data type enumerant for which a record element applies.
{}: The body of the variant record.
[]: The variant part of the variant record.

11.2.9.2 SRFT parameters

11.2.9.2.1 EC_Parameters

This non-object 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;
}
```

11.2.9.2.2 LCC_Parameters

This non-object 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;
}
```

11.2.9.2.3 LSR_2D_Parameters

This non-object data type specifies the parameters that correspond to SRFT LOCAL_SPACE_RECTANGULAR_2D.

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

11.2.9.2.4 LSR_3D_Parameters

This non-object data type specifies the parameters that correspond to SRFT LOCAL_SPACE_RECTANGULAR_3D.

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

11.2.9.2.5 Local_Tangent_Parameters

This non-object 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;
}
```

11.2.9.2.6 LTSE_Parameters

This non-object 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;
}
```

11.2.9.2.7 LCE_3D_Parameters

This non-object 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;
}
```

11.2.9.2.8 M_Parameters

This non-object 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;
}
```

11.2.9.2.9 Oblique_Mercator_Parameters

This non-object 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;
}
```

11.2.9.2.10 PS_Parameters

This non-object 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;
   false_northing                 Long_Float;
}
```

11.2.9.2.11 SRFS_Code_Info

This Variant Record Data Type specifies an arbitrary SRFS_Code with its associated SRFS member code. The record element SRFSM_unspecified shall be set to zero (unspecified) when the selector value is SRFS_UNDEFINED.

```
SRFS_Code_Info ::= ( srfs_code   SRFS_Code )
{
   [
   SRFS_UNSPECIFIED:
     SRFSM_unspecified            Integer;
   SRFS_ALABAMA_SPCS:
     SRFSM_alabama_spcs           SRFSM_Alabama_SPCS_Code;
   SRFS_GTRS_GLOBAL_COORDINATE_SYSTEM:
     SRFSM_gtrs_global_coordinate_system
                                  SRFSM_GTRS_Global_Coordinate_System_Code;
   SRFS_JAPAN_RECTANGULAR_PLANE_CS:
     SRFSM_japan_rectangular_plane_cs
                                  SRFSM_Japan_Rectangular_Plane_CS_Code;
   SRFS_LAMBERT_NTF:
     SRFSM_lambert_ntf            SRFSM_Lambert_NTF_Code;
   SRFS_UNIVERSAL_POLAR_STEREOGRAPHIC:
     SRFSM_universal_polar_stereographic
                                  SRFSM_Universal_Polar_Stereographic_Code;
   SRFS_UNIVERSAL_TRANSVERSE_MERCATOR:
     SRFSM_universal_transverse_mercator
                                  SRFSM_Universal_Transverse_Mercator_Code;
   SRFS_WISCONSIN_SPCS:
     SRFSM_wisconsin_spcs         SRFSM_Wisconsin_SPCS_Code;
   ]
}
```

11.2.9.2.12 TM_Parameters

This non-object 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;
}
```

11.2.9.3 ORM transformation parameters

11.2.9.3.1 ORM_Transformation_2D_Parameters

This non-object data type represents a 2D ORM four-parameter transformation as specified in 7.3.3.

```
ORM_Transformation_2D_Parameters ::= {
   delta_x                        Long_Float;
   delta_y                        Long_Float;
   omega                          Long_Float;
   delta_s                        Long_Float;
}
```

The valid range in radians for values of omega is (-2π, 2π). The valid range for delta_s is greater than -1.

11.2.9.3.2 ORM_Transformation_3D_Parameters

This non-object data type represents a 3D ORM seven-parameter transformation as specified in 7.3.2.

```
ORM_Transformation_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_s                        Long_Float;
}
```

The valid range in radians for values omega_1, omega_2, and omega_3 is (-2π, 2π). The valid range for delta_s is greater than -1.

11.3 Object classes

11.3.1 Introduction

SRF objects 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 remaining functionality is provided in concrete class and function specifications. 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 methods manipulate internal data (object state) and any input parameters passed in. The success condition is a nominal behaviour of all methods and is not listed within the error conditions element. The success condition is associated with `Status_Code SUCCESS`.

EXAMPLE 1 In Table 11.13, the phrase "this SRF" refers to the internal state of an instance of a concrete class subclassed (directly or indirectly) from the abstract class specified in the table. In particular, the abstract method `GetORMCode` "Outputs the `ORM_Code` and the `RT_Code` of this SRF", and shows "Inputs: none".

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. Under an error condition, output values are undefined. When several error conditions apply to a method invocation, the first error condition detected by an implementation shall be presented as the method status. The error conditions applicable to a method invocation are the common error conditions specified in 11.3.2 and the additional error conditions specified in the class specification table for the method and any language-binding specific error conditions applicable to the method.

A language binding mechanism for presentation of method status shall support the association of a unique error `Status_Code` (11.2.7.11)

EXAMPLE 2 If a language binding supports exception handling and if a language binding uses that mechanism to present method failure, then an exception object method that returns the corresponding `Status_Code` would satisfy this requirement.

11.3.2 Class specification format

Class data types are specified in tables in Table 11.5 through Table 11.44 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 or** **Private 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 object is implicitly an input and is not additionally listed in this element. The Create method of an object class is an exception. The Create method of an object class depends only on its explicit input parameters.