Sage Reference Manual: Manifolds

Release 7.1

The Sage Development Team

March 20, 2016

1 Topological Manifolds **3**
1.1 Topological Manifolds 3
1.2 Subsets of Topological Manifolds 21
1.3 Manifold Structures 32
1.4 Points of Topological Manifolds 33
1.5 Coordinate Charts 42

2 Indices and Tables **61**

Bibliography **63**

This is the Sage implementation of manifolds resulting from the SageManifolds project. This section describes only the "manifold" part of SageManifolds; the pure algebraic part is described in the section *Tensors on free modules of finite rank*.

More documentation (in particular example worksheets) can be found here.

CHAPTER

ONE

TOPOLOGICAL MANIFOLDS

1.1 Topological Manifolds

Given a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$) and a non-negative integer n, a *topological manifold of dimension n over K* is a topological space M such that

- M is a Hausdorff space,
- M is second countable,
- every point in M has a neighborhood homeomorphic to K^n.

Topological manifolds are implemented via the class `TopologicalManifold`. Open subsets of topological manifolds are also implemented via `TopologicalManifold`, since they are topological manifolds by themselves.

In the current setting, topological manifolds are mostly described by means of charts (see `Chart`).

`TopologicalManifold` serves as a base class for more specific manifold classes.

The user interface is provided by the generic function `Manifold()`, with with the argument `structure` set to `'topological'`.

Example 1: the 2-sphere as a topological manifold of dimension 2 over $\mathbf{R}$

One starts by declaring S^2 as a 2-dimensional topological manifold:

```
sage: M = Manifold(2, 'S^2', structure='topological')
sage: M
2-dimensional topological manifold S^2
```

Since the base topological field has not been specified in the argument list of `Manifold`, $\mathbf{R}$ is assumed:

```
sage: M.base_field()
Real Field with 53 bits of precision
sage: dim(M)
2
```

Let us consider the complement of a point, the "North pole" say; this is an open subset of S^2, which we call U:

```
sage: U = M.open_subset('U'); U
Open subset U of the 2-dimensional topological manifold S^2
```

A standard chart on U is provided by the stereographic projection from the North pole to the equatorial plane:

```
sage: stereoN.<x,y> = U.chart(); stereoN
Chart (U, (x, y))
```

Thanks to the operator `<x,y>` on the left-hand side, the coordinates declared in a chart (here x and y), are accessible by their names; they are Sage's symbolic variables:

```
sage: y
y
sage: type(y)
<type 'sage.symbolic.expression.Expression'>
```

The South pole is the point of coordinates $(x, y) = (0, 0)$ in the above chart:

```
sage: S = U.point((0,0), chart=stereoN, name='S'); S
Point S on the 2-dimensional topological manifold S^2
```

Let us call V the open subset that is the complement of the South pole and let us introduce on it the chart induced by the stereographic projection from the South pole to the equatorial plane:

```
sage: V = M.open_subset('V'); V
Open subset V of the 2-dimensional topological manifold S^2
sage: stereoS.<u,v> = V.chart(); stereoS
Chart (V, (u, v))
```

The North pole is the point of coordinates $(u, v) = (0, 0)$ in this chart:

```
sage: N = V.point((0,0), chart=stereoS, name='N'); N
Point N on the 2-dimensional topological manifold S^2
```

To fully construct the manifold, we declare that it is the union of U and V:

```
sage: M.declare_union(U,V)
```

and we provide the transition map between the charts `stereoN` = $(U, (x, y))$ and `stereoS` = $(V, (u, v))$, denoting by W the intersection of U and V (W is the subset of U defined by $x^2 + y^2 \neq 0$, as well as the subset of V defined by $u^2 + v^2 \neq 0$):

```
sage: stereoN_to_S = stereoN.transition_map(stereoS, [x/(x^2+y^2), y/(x^2+y^2)],
....:                           intersection_name='W', restrictions1= x^2+y^2!=0,
....:                                                  restrictions2= u^2+v^2!=0)
sage: stereoN_to_S
Change of coordinates from Chart (W, (x, y)) to Chart (W, (u, v))
sage: stereoN_to_S.display()
u = x/(x^2 + y^2)
v = y/(x^2 + y^2)
```

We give the name `W` to the Python variable representing $W = U \cap V$:

```
sage: W = U.intersection(V)
```

The inverse of the transition map is computed by the method `sage.manifolds.chart.CoordChange.inverse()`:

```
sage: stereoN_to_S.inverse()
Change of coordinates from Chart (W, (u, v)) to Chart (W, (x, y))
sage: stereoN_to_S.inverse().display()
x = u/(u^2 + v^2)
y = v/(u^2 + v^2)
```

At this stage, we have four open subsets on S^2:

```
sage: M.list_of_subsets()
[2-dimensional topological manifold S^2,
 Open subset U of the 2-dimensional topological manifold S^2,
 Open subset V of the 2-dimensional topological manifold S^2,
 Open subset W of the 2-dimensional topological manifold S^2]
```

W is the open subset that is the complement of the two poles:

```
sage: N in W or S in W
False
```

The North pole lies in V and the South pole in U:

```
sage: N in V, N in U
(True, False)
sage: S in U, S in V
(True, False)
```

The manifold's (user) atlas contains four charts, two of them being restrictions of charts to a smaller domain:

```
sage: M.atlas()
[Chart (U, (x, y)), Chart (V, (u, v)),
 Chart (W, (x, y)), Chart (W, (u, v))]
```

Let us consider the point of coordinates $(1, 2)$ in the chart `stereoN`:

```
sage: p = M.point((1,2), chart=stereoN, name='p'); p
Point p on the 2-dimensional topological manifold S^2
sage: p.parent()
2-dimensional topological manifold S^2
sage: p in W
True
```

The coordinates of p in the chart `stereoS` are computed by letting the chart act on the point:

```
sage: stereoS(p)
(1/5, 2/5)
```

Given the definition of p, we have of course:

```
sage: stereoN(p)
(1, 2)
```

Similarly:

```
sage: stereoS(N)
(0, 0)
sage: stereoN(S)
(0, 0)
```

Example 2: the Riemann sphere as a topological manifold of dimension 1 over $\mathbb{C}$

We declare the Riemann sphere $\mathbb{C}^*$ as a 1-dimensional topological manifold over $\mathbb{C}$:

```
sage: M = Manifold(1, 'C*', structure='topological', field='complex'); M
Complex 1-dimensional topological manifold C*
```

We introduce a first open subset, which is actually $\mathbf{C} = \mathbf{C}^* \setminus \{\infty\}$ if we interpret $\mathbf{C}^*$ as the Alexandroff one-point compactification of $\mathbf{C}$:

```
sage: U = M.open_subset('U')
```

A natural chart on U is then nothing but the identity map of $\mathbf{C}$, hence we denote the associated coordinate by z:

```
sage: Z.<z> = U.chart()
```

The origin of the complex plane is the point of coordinate $z = 0$:

```
sage: O = U.point((0,), chart=Z, name='O'); O
Point O on the Complex 1-dimensional topological manifold C*
```

Another open subset of $\mathbf{C}^*$ is $V = \mathbf{C}^* \setminus \{O\}$:

```
sage: V = M.open_subset('V')
```

We define a chart on V such that the point at infinity is the point of coordinate 0 in this chart:

```
sage: W.<w> = V.chart(); W
Chart (V, (w,))
sage: inf = M.point((0,), chart=W, name='inf', latex_name=r'\infty')
sage: inf
Point inf on the Complex 1-dimensional topological manifold C*
```

To fully construct the Riemann sphere, we declare that it is the union of U and V:

```
sage: M.declare_union(U,V)
```

and we provide the transition map between the two charts as $w = 1/z$ on $A = U \cap V$:

```
sage: Z_to_W = Z.transition_map(W, 1/z, intersection_name='A',
....:                           restrictions1= z!=0, restrictions2= w!=0)
sage: Z_to_W
Change of coordinates from Chart (A, (z,)) to Chart (A, (w,))
sage: Z_to_W.display()
w = 1/z
sage: Z_to_W.inverse()
Change of coordinates from Chart (A, (w,)) to Chart (A, (z,))
sage: Z_to_W.inverse().display()
z = 1/w
```

Let consider the complex number i as a point of the Riemann sphere:

```
sage: i = M((I,), chart=Z, name='i'); i
Point i on the Complex 1-dimensional topological manifold C*
```

Its coordinates w.r.t. the charts `Z` and `W` are:

```
sage: Z(i)
(I,)
sage: W(i)
(-I,)
```

and we have:

```
sage: i in U
True
sage: i in V
True
```

The following subsets and charts have been defined:

```
sage: M.list_of_subsets()
[Open subset A of the Complex 1-dimensional topological manifold C*,
 Complex 1-dimensional topological manifold C*,
 Open subset U of the Complex 1-dimensional topological manifold C*,
 Open subset V of the Complex 1-dimensional topological manifold C*]
sage: M.atlas()
[Chart (U, (z,)), Chart (V, (w,)), Chart (A, (z,)), Chart (A, (w,))]
```

AUTHORS:

- Eric Gourgoulhon (2015): initial version
- Travis Scrimshaw (2015): structure described via `TopologicalStructure` or `RealTopologicalStructure`

REFERENCES:

`sage.manifolds.manifold.`**`Manifold`**(*dim*, *name*, *latex_name=None*, *field='real'*, *structure='smooth'*, *start_index=0*, ***extra_kwds*)

Construct a manifold of a given type over a topological field K.

Given a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$) and a non-negative integer n, a *topological manifold of dimension n over K* is a topological space M such that

- M is a Hausdorff space,
- M is second countable, and
- every point in M has a neighborhood homeomorphic to K^n.

A *real manifold* is a manifold over $\mathbf{R}$. A *differentiable* (resp. *smooth*, resp. *analytic*) is a real manifold such that all transition maps are *differentiable* (resp. *smooth*, resp. *analytic*).

INPUT:

- `dim` – positive integer; dimension of the manifold
- `name` – string; name (symbol) given to the manifold
- `latex_name` – (default: `None`) string; LaTeX symbol to denote the manifold; if none are provided, it is set to `name`
- `field` – (default: `'real'`) field K on which the manifold is defined; allowed values are
  - `'real'` or an object of type `RealField` (e.g. `RR`) for a manifold over $\mathbf{R}$
  - `'complex'` or an object of type `ComplexField` (e.g. `CC`) for a manifold over $\mathbf{C}$
  - an object in the category of topological fields (see `Fields` and `TopologicalSpaces`) for other types of manifolds
- `structure` – (default: `'smooth'`) to specify the structure or type of manifold; allowed values are
  - `'topological'` or `'top'` for a topological manifold
  - `'differentiable'` or `'diff'` for a differentiable manifold

  - 'smooth' for a smooth manifold
  - 'analytic' for an analytic manifold

- start_index – (default: 0) integer; lower value of the range of indices used for "indexed objects" on the manifold, e.g. coordinates in a chart
- extra_kwds – keywords meaningful only for some specific types of manifolds

OUTPUT:

- a manifold of the specified type

EXAMPLES:

A 3-dimensional real topological manifold:

```
sage: M = Manifold(3, 'M', structure='topological'); M
3-dimensional topological manifold M
```

Given the default value of the parameter field, the above is equivalent to:

```
sage: M = Manifold(3, 'M', structure='topological', field='real'); M
3-dimensional topological manifold M
```

A complex topological manifold:

```
sage: M = Manifold(3, 'M', structure='topological', field='complex'); M
Complex 3-dimensional topological manifold M
```

A topological manifold over $\mathbf{Q}$:

```
sage: M = Manifold(3, 'M', structure='topological', field=QQ); M
3-dimensional topological manifold M over the Rational Field
```

See the documentation of class TopologicalManifold for more detailed examples.

Uniqueness of manifold objects

Suppose we construct a manifold named M:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
```

At some point, we change our mind and would like to restart with a new manifold, using the same name M and keeping the previous manifold for reference:

```
sage: M_old = M  # for reference
sage: M = Manifold(2, 'M', structure='topological')
```

This results in a brand new object:

```
sage: M.atlas()
[]
```

The object M_old is intact:

```
sage: M_old.atlas()
[Chart (M, (x, y))]
```

Both objects have the same display:

```
sage: M
2-dimensional topological manifold M
sage: M_old
2-dimensional topological manifold M
```

but they are different:

```
sage: M != M_old
True
```

Let us introduce a chart on `M`, using the same coordinate symbols as for `M_old`:

```
sage: X.<x,y> = M.chart()
```

The charts are displayed in the same way:

```
sage: M.atlas()
[Chart (M, (x, y))]
sage: M_old.atlas()
[Chart (M, (x, y))]
```

but they are actually different:

```
sage: M.atlas()[0] != M_old.atlas()[0]
True
```

Moreover, the two manifolds `M` and `M_old` are still considered distinct:

```
sage: M != M_old
True
```

This reflects the fact that the equality of manifold objects holds only for identical objects, i.e. one has `M1 == M2` if, and only if, `M1 is M2`. Actually, the manifold classes inherit from `WithEqualityById`:

```
sage: isinstance(M, sage.misc.fast_methods.WithEqualityById)
True
```

class sage.manifolds.manifold.**TopologicalManifold**(*n*, *name*, *field*, *structure*, *ambient=None*, *latex_name=None*, *start_index=0*, *category=None*, *unique_tag=None*)

Bases: sage.manifolds.subset.ManifoldSubset

Topological manifold over a topological field K.

Given a topological field K (in most applications, $K = \mathbf{R}$ or $K = \mathbf{C}$) and a non-negative integer n, a *topological manifold of dimension n over K* is a topological space M such that

- M is a Hausdorff space,
- M is second countable, and
- every point in M has a neighborhood homeomorphic to K^n.

This is a Sage *parent* class, the corresponding *element* class being `ManifoldPoint`.

INPUT:

- `n` – positive integer; dimension of the manifold
- `name` – string; name (symbol) given to the manifold
- `field` – field K on which the manifold is defined; allowed values are

  - 'real' or an object of type RealField (e.g., RR) for a manifold over $\mathbf{R}$
  - 'complex' or an object of type ComplexField (e.g., CC) for a manifold over $\mathbf{C}$
  - an object in the category of topological fields (see Fields and TopologicalSpaces) for other types of manifolds
- structure – manifold structure (see TopologicalStructure or RealTopologicalStructure)
- ambient – (default: None) if not None, must be a topological manifold; the created object is then an open subset of ambient
- latex_name – (default: None) string; LaTeX symbol to denote the manifold; if none are provided, it is set to name
- start_index – (default: 0) integer; lower value of the range of indices used for "indexed objects" on the manifold, e.g., coordinates in a chart
- category – (default: None) to specify the category; if None, Manifolds(field) is assumed (see the category Manifolds)
- unique_tag – (default: None) tag used to force the construction of a new object when all the other arguments have been used previously (without unique_tag, the UniqueRepresentation behavior inherited from ManifoldSubset would return the previously constructed object corresponding to these arguments)

EXAMPLES:

A 4-dimensional topological manifold (over $\mathbf{R}$):

```
sage: M = Manifold(4, 'M', latex_name=r'\mathcal{M}', structure='topological')
sage: M
4-dimensional topological manifold M
sage: latex(M)
\mathcal{M}
sage: type(M)
<class 'sage.manifolds.manifold.TopologicalManifold_with_category'>
sage: M.base_field()
Real Field with 53 bits of precision
sage: dim(M)
4
```

The input parameter start_index defines the range of indices on the manifold:

```
sage: M = Manifold(4, 'M', structure='topological')
sage: list(M.irange())
[0, 1, 2, 3]
sage: M = Manifold(4, 'M', structure='topological', start_index=1)
sage: list(M.irange())
[1, 2, 3, 4]
sage: list(Manifold(4, 'M', structure='topological', start_index=-2).irange())
[-2, -1, 0, 1]
```

A complex manifold:

```
sage: N = Manifold(3, 'N', structure='topological', field='complex'); N
Complex 3-dimensional topological manifold N
```

A manifold over $\mathbf{Q}$:

```
sage: N = Manifold(6, 'N', structure='topological', field=QQ); N
6-dimensional topological manifold N over the Rational Field
```

A manifold over $\mathbf{Q}_5$, the field of 5-adic numbers:

```
sage: N = Manifold(2, 'N', structure='topological', field=Qp(5)); N
2-dimensional topological manifold N over the 5-adic Field with capped
 relative precision 20
```

A manifold is a Sage *parent* object, in the category of topological manifolds over a given topological field (see `Manifolds`):

```
sage: isinstance(M, Parent)
True
sage: M.category()
Category of manifolds over Real Field with 53 bits of precision
sage: from sage.categories.manifolds import Manifolds
sage: M.category() is Manifolds(RR)
True
sage: M.category() is Manifolds(M.base_field())
True
sage: M in Manifolds(RR)
True
sage: N in Manifolds(Qp(5))
True
```

The corresponding Sage *elements* are points:

```
sage: X.<t, x, y, z> = M.chart()
sage: p = M.an_element(); p
Point on the 4-dimensional topological manifold M
sage: p.parent()
4-dimensional topological manifold M
sage: M.is_parent_of(p)
True
sage: p in M
True
```

The manifold's points are instances of class `ManifoldPoint`:

```
sage: isinstance(p, sage.manifolds.point.ManifoldPoint)
True
```

Since an open subset of a topological manifold M is itself a topological manifold, open subsets of M are instances of the class `TopologicalManifold`:

```
sage: U = M.open_subset('U'); U
Open subset U of the 4-dimensional topological manifold M
sage: isinstance(U, sage.manifolds.manifold.TopologicalManifold)
True
sage: U.base_field() == M.base_field()
True
sage: dim(U) == dim(M)
True
sage: U.category()
Join of Category of subobjects of sets and Category of manifolds over
 Real Field with 53 bits of precision
```

The manifold passes all the tests of the test suite relative to its category:

```
sage: TestSuite(M).run()
```

See also:

sage.manifolds.manifold

atlas()

Return the list of charts that have been defined on the manifold.

EXAMPLES:

Let us consider $\mathbf{R}^2$ as a 2-dimensional manifold:

```
sage: M = Manifold(2, 'R^2', structure='topological')
```

Immediately after the manifold creation, the atlas is empty, since no chart has been defined yet:

```
sage: M.atlas()
[]
```

Let us introduce the chart of Cartesian coordinates:

```
sage: c_cart.<x,y> = M.chart()
sage: M.atlas()
[Chart (R^2, (x, y))]
```

The complement of the half line $\{y = 0, x \geq 0\}$:

```
sage: U = M.open_subset('U', coord_def={c_cart: (y!=0,x<0)})
sage: U.atlas()
[Chart (U, (x, y))]
sage: M.atlas()
[Chart (R^2, (x, y)), Chart (U, (x, y))]
```

Spherical (polar) coordinates on U:

```
sage: c_spher.<r, ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: U.atlas()
[Chart (U, (x, y)), Chart (U, (r, ph))]
sage: M.atlas()
[Chart (R^2, (x, y)), Chart (U, (x, y)), Chart (U, (r, ph))]
```

See also:

top_charts()

base_field()

Return the field on which the manifold is defined.

OUTPUT:

- a topological field

EXAMPLES:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: M.base_field()
Real Field with 53 bits of precision
sage: M = Manifold(3, 'M', structure='topological', field='complex')
sage: M.base_field()
Complex Field with 53 bits of precision
sage: M = Manifold(3, 'M', structure='topological', field=QQ)
sage: M.base_field()
Rational Field
```

base_field_type()

Return the type of topological field on which the manifold is defined.

OUTPUT:

- a string describing the field, with three possible values:
  - `'real'` for the real field $\mathbf{R}$
  - `'complex'` for the complex field $\mathbf{C}$
  - `'neither_real_nor_complex'` for a field different from $\mathbf{R}$ and $\mathbf{C}$

EXAMPLES:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: M.base_field_type()
'real'
sage: M = Manifold(3, 'M', structure='topological', field='complex')
sage: M.base_field_type()
'complex'
sage: M = Manifold(3, 'M', structure='topological', field=QQ)
sage: M.base_field_type()
'neither_real_nor_complex'
```

chart(*coordinates=''*, *names=None*)

Define a chart, the domain of which is the manifold.

A *chart* is a pair (U, φ), where U is the current manifold and $\varphi : U \to V \subset K^n$ is a homeomorphism from U to an open subset V of K^n, K being the field on which the manifold is defined.

The components $(x^1, \ldots, x^n)$ of φ, defined by $\varphi(p) = (x^1(p), \ldots, x^n(p)) \in K^n$ for any point $p \in U$, are called the *coordinates* of the chart (U, φ).

See `Chart` for a complete documentation.

INPUT:

- `coordinates` – (default: `''` (empty string)) string defining the coordinate symbols and ranges, see below
- `names` – (default: `None`) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator `<,>` is used)

The coordinates declared in the string `coordinates` are separated by `' '` (whitespace) and each coordinate has at most three fields, separated by a colon (`':'`):

1. The coordinate symbol (a letter or a few letters).
2. (optional, only for manifolds over $\mathbf{R}$) The interval I defining the coordinate range: if not provided, the coordinate is assumed to span all $\mathbf{R}$; otherwise I must be provided in the form `(a,b)` (or equivalently `]a,b[`) The bounds `a` and `b` can be `+/-Infinity`, `Inf`, `infinity`, `inf` or `oo`. For *singular* coordinates, non-open intervals such as `[a,b]` and `(a,b]` (or equivalently `]a,b]`) are allowed. Note that the interval declaration must not contain any space character.
3. (optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

The order of the fields 2 and 3 does not matter and each of them can be omitted. If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of the backslash character (see examples below). If no interval range and no LaTeX spelling is to be provided for any coordinate, the argument `coordinates` can be omitted when the shortcut operator `<,>` is used via Sage preparser (see examples below).

OUTPUT:

- the created chart, as an instance of `Chart` or of the subclass `RealChart` for manifolds over $\mathbf{R}$.

EXAMPLES:

Chart on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: X = U.chart('x y'); X
Chart (U, (x, y))
sage: X[0]
x
sage: X[1]
y
sage: X[:]
(x, y)
```

The declared coordinates are not known at the global level:

```
sage: y
Traceback (most recent call last):
...
NameError: name 'y' is not defined
```

They can be recovered by the operator `[:]` applied to the chart:

```
sage: (x, y) = X[:]
sage: y
y
sage: type(y)
<type 'sage.symbolic.expression.Expression'>
```

But a shorter way to proceed is to use the operator <,> in the left-hand side of the chart declaration (there is then no need to pass the string 'x y' to chart()):

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: X.<x,y> = U.chart(); X
Chart (U, (x, y))
```

Indeed, the declared coordinates are then known at the global level:

```
sage: y
y
sage: (x,y) == X[:]
True
```

Actually the instruction `X.<x,y> = U.chart()` is equivalent to the combination of the two instructions `X = U.chart('x y')` and `(x,y) = X[:]`.

See the documentation of class `Chart` for more examples, especially regarding the coordinates ranges and restrictions.

coord_change(*chart1*, *chart2*)

Return the change of coordinates (transition map) between two charts defined on the manifold.

The change of coordinates must have been defined previously, for instance by the method `transition_map()`.

INPUT:

- `chart1` – chart 1
- `chart2` – chart 2

OUTPUT:

- instance of `CoordChange` representing the transition map from chart 1 to chart 2

EXAMPLES:

Change of coordinates on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: c_xy.transition_map(c_uv, (x+y, x-y)) # defines the coord. change
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
sage: M.coord_change(c_xy, c_uv) # returns the coord. change defined above
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
```

coord_changes()

Return the changes of coordinates (transition maps) defined on subsets of the manifold.

OUTPUT:

- dictionary of changes of coordinates, with pairs of charts as keys

EXAMPLES:

Various changes of coordinates on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, [x+y, x-y])
sage: M.coord_changes()
{(Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))}
sage: uv_to_xy = xy_to_uv.inverse()
sage: M.coord_changes()  # random (dictionary output)
{(Chart (M, (u, v)),
  Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y)),
 (Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))}
sage: c_rs.<r,s> = M.chart()
sage: uv_to_rs = c_uv.transition_map(c_rs, [-u+2*v, 3*u-v])
sage: M.coord_changes()  # random (dictionary output)
{(Chart (M, (u, v)),
  Chart (M, (r, s))): Change of coordinates from Chart (M, (u, v)) to Chart (M, (r, s)),
 (Chart (M, (u, v)),
  Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y)),
 (Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))}
sage: xy_to_rs = uv_to_rs * xy_to_uv
sage: M.coord_changes()  # random (dictionary output)
{(Chart (M, (u, v)),
  Chart (M, (r, s))): Change of coordinates from Chart (M, (u, v)) to Chart (M, (r, s)),
 (Chart (M, (u, v)),
  Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y)),
 (Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v)),
 (Chart (M, (x, y)),
  Chart (M, (r, s))): Change of coordinates from Chart (M, (x, y)) to Chart (M, (r, s))}
```

default_chart()

Return the default chart defined on the manifold.

Unless changed via `set_default_chart()`, the *default chart* is the first one defined on a subset of the manifold (possibly itself).

OUTPUT:

- instance of `Chart` representing the default chart

EXAMPLES:

Default chart on a 2-dimensional manifold and on some subsets:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: M.chart('x y')
Chart (M, (x, y))
sage: M.chart('u v')
Chart (M, (u, v))
sage: M.default_chart()
Chart (M, (x, y))
sage: A = M.open_subset('A')
sage: A.chart('t z')
Chart (A, (t, z))
sage: A.default_chart()
Chart (A, (t, z))
```

dim()

Return the dimension of the manifold over its base field.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: M.dimension()
2
```

A shortcut is `dim()`:

```
sage: M.dim()
2
```

The Sage global function `dim` can also be used:

```
sage: dim(M)
2
```

dimension()

Return the dimension of the manifold over its base field.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: M.dimension()
2
```

A shortcut is `dim()`:

```
sage: M.dim()
2
```

The Sage global function `dim` can also be used:

```
sage: dim(M)
2
```

get_chart(*coordinates*, *domain=None*)

Get a chart from its coordinates.

The chart must have been previously created by the method `chart()`.

INPUT:

- `coordinates` – single string composed of the coordinate symbols separated by a space
- `domain` – (default: `None`) string containing the name of the chart's domain, which must be a subset of the current manifold; if `None`, the current manifold is assumed

OUTPUT:

- instance of `Chart` (or of the subclass `RealChart`) representing the chart corresponding to the above specifications

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: M.get_chart('x y')
Chart (M, (x, y))
sage: M.get_chart('x y') is X
True
sage: U = M.open_subset('U', coord_def={X: (y!=0,x<0)})
sage: Y.<r, ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (x, y)), Chart (U, (r, ph))]
sage: M.get_chart('x y', domain='U')
Chart (U, (x, y))
sage: M.get_chart('x y', domain='U') is X.restrict(U)
True
sage: U.get_chart('r ph')
Chart (U, (r, ph))
sage: M.get_chart('r ph', domain='U')
Chart (U, (r, ph))
sage: M.get_chart('r ph', domain='U') is Y
True
```

index_generator(*nb_indices*)

Generator of index series.

INPUT:

- `nb_indices` – number of indices in a series

OUTPUT:

- an iterable index series for a generic component with the specified number of indices

EXAMPLES:

Indices on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological', start_index=1)
sage: for ind in M.index_generator(2):
....:     print ind
....:
(1, 1)
(1, 2)
(2, 1)
(2, 2)
```

Loops can be nested:

```
sage: for ind1 in M.index_generator(2):
....:     print ind1, " : ",
....:     for ind2 in M.index_generator(2):
....:         print ind2,
....:     print ""
....:
(1, 1)  :  (1, 1) (1, 2) (2, 1) (2, 2)
(1, 2)  :  (1, 1) (1, 2) (2, 1) (2, 2)
(2, 1)  :  (1, 1) (1, 2) (2, 1) (2, 2)
(2, 2)  :  (1, 1) (1, 2) (2, 1) (2, 2)
```

irange(*start=None*)

Single index generator.

INPUT:

- `start` – (default: `None`) initial value i_0 of the index; if none are provided, the value returned by `start_index()` is assumed

OUTPUT:

- an iterable index, starting from i_0 and ending at $i_0 + n - 1$, where n is the manifold's dimension

EXAMPLES:

Index range on a 4-dimensional manifold:

```
sage: M = Manifold(4, 'M', structure='topological')
sage: for i in M.irange():
....:     print i,
....:
0 1 2 3
sage: for i in M.irange(2):
....:     print i,
....:
2 3
sage: list(M.irange())
[0, 1, 2, 3]
```

Index range on a 4-dimensional manifold with starting index=1:

```
sage: M = Manifold(4, 'M', structure='topological', start_index=1)
sage: for i in M.irange():
....:     print i,
....:
1 2 3 4
sage: for i in M.irange(2):
....:    print i,
....:
2 3 4
```

In general, one has always:

```
sage: M.irange().next() == M.start_index()
True
```

is_manifestly_coordinate_domain()

Return `True` if the manifold is known to be the domain of some coordinate chart and `False` otherwise.

If `False` is returned, either the manifold cannot be the domain of some coordinate chart or no such chart has been declared yet.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: X.<x,y> = U.chart()
sage: U.is_manifestly_coordinate_domain()
True
sage: M.is_manifestly_coordinate_domain()
False
sage: Y.<u,v> = M.chart()
sage: M.is_manifestly_coordinate_domain()
True
```

is_open()

Return if `self` is an open set.

In the present case (manifold or open subset of it), always return `True`.

TEST:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: M.is_open()
True
```

open_subset(*name*, *latex_name=None*, *coord_def={}*)

Create an open subset of the manifold.

An open subset is a set that is (i) included in the manifold and (ii) open with respect to the manifold's topology. It is a topological manifold by itself. Hence the returned object is an instance of `TopologicalManifold`.

INPUT:

- `name` – name given to the open subset
- `latex_name` – (default: `None`) LaTeX symbol to denote the subset; if none are provided, it is set to `name`
- `coord_def` – (default: {}) definition of the subset in terms of coordinates; `coord_def` must a be dictionary with keys charts on the manifold and values the symbolic expressions formed by the coordinates to define the subset

OUTPUT:

- the open subset, as an instance of `TopologicalManifold`

EXAMPLES:

Creating an open subset of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.open_subset('A'); A
Open subset A of the 2-dimensional topological manifold M
```

As an open subset of a topological manifold, `A` is itself a topological manifold, on the same topological field and of the same dimension as `M`:

```
sage: isinstance(A, sage.manifolds.manifold.TopologicalManifold)
True
sage: A.base_field() == M.base_field()
True
sage: dim(A) == dim(M)
True
```

```
sage: A.category() is M.category().Subobjects()
True
```

Creating an open subset of A:

```
sage: B = A.open_subset('B'); B
Open subset B of the 2-dimensional topological manifold M
```

We have then:

```
sage: A.subsets()  # random (set output)
{Open subset B of the 2-dimensional topological manifold M,
 Open subset A of the 2-dimensional topological manifold M}
sage: B.is_subset(A)
True
sage: B.is_subset(M)
True
```

Defining an open subset by some coordinate restrictions: the open unit disk in $\mathbf{R}^2$:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: U = M.open_subset('U', coord_def={c_cart: x^2+y^2<1}); U
Open subset U of the 2-dimensional topological manifold R^2
```

Since the argument coord_def has been set, U is automatically provided with a chart, which is the restriction of the Cartesian one to U:

```
sage: U.atlas()
[Chart (U, (x, y))]
```

Therefore, one can immediately check whether a point belongs to U:

```
sage: M.point((0,0)) in U
True
sage: M.point((1/2,1/3)) in U
True
sage: M.point((1,2)) in U
False
```

set_default_chart(*chart*)

Changing the default chart on self.

INPUT:

- chart – a chart (must be defined on some subset self)

EXAMPLES:

Charts on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: M.default_chart()
Chart (M, (x, y))
sage: M.set_default_chart(c_uv)
sage: M.default_chart()
Chart (M, (u, v))
```

start_index()

Return the first value of the index range used on the manifold.

This is the parameter `start_index` passed at the construction of the manifold.

OUTPUT:

- the integer i_0 such that all indices of indexed objects on the manifold range from i_0 to $i_0 + n - 1$, where n is the manifold's dimension

EXAMPLES:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: M.start_index()
0
sage: M = Manifold(3, 'M', structure='topological', start_index=1)
sage: M.start_index()
1
```

top_charts()

Return the list of charts defined on subsets of the current manifold that are not subcharts of charts on larger subsets.

OUTPUT:

- list of charts defined on open subsets of the manifold but not on larger subsets

EXAMPLES:

Charts on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: U = M.open_subset('U', coord_def={X: x>0})
sage: Y.<u,v> = U.chart()
sage: M.top_charts()
[Chart (M, (x, y)), Chart (U, (u, v))]
```

Note that the (user) atlas contains one more chart: `(U, (x,y))`, which is not a "top" chart:

```
sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (x, y)), Chart (U, (u, v))]
```

See also:

`atlas()` for the complete list of charts defined on the manifold.

1.2 Subsets of Topological Manifolds

The class `ManifoldSubset` implements generic subsets of a topological manifold. Open subsets are implemented by the class `TopologicalManifold` (since an open subset of a manifold is a manifold by itself), which inherits from `ManifoldSubset`.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015): initial version
- Travis Scrimshaw (2015): review tweaks; removal of facade parents

REFERENCES:

- [Lee11] J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)

EXAMPLES:

Two subsets on a manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A'); a
Subset A of the 2-dimensional topological manifold M
sage: b = M.subset('B'); b
Subset B of the 2-dimensional topological manifold M
sage: M.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset B of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M]
```

The intersection of the two subsets:

```
sage: c = a.intersection(b); c
Subset A_inter_B of the 2-dimensional topological manifold M
```

Their union:

```
sage: d = a.union(b); d
Subset A_union_B of the 2-dimensional topological manifold M
```

Lists of subsets after the above operations:

```
sage: M.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset A_inter_B of the 2-dimensional topological manifold M,
 Subset A_union_B of the 2-dimensional topological manifold M,
 Subset B of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M]
sage: a.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset A_inter_B of the 2-dimensional topological manifold M]
sage: c.list_of_subsets()
[Subset A_inter_B of the 2-dimensional topological manifold M]
sage: d.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset A_inter_B of the 2-dimensional topological manifold M,
 Subset A_union_B of the 2-dimensional topological manifold M,
 Subset B of the 2-dimensional topological manifold M]
```

class sage.manifolds.subset.**ManifoldSubset**(*manifold*, *name*, *latex_name=None*, *category=None*)

Bases: sage.structure.unique_representation.UniqueRepresentation, sage.structure.parent.Parent

Subset of a topological manifold.

The class ManifoldSubset inherits from the generic class Parent. The corresponding element class is ManifoldPoint.

Note that open subsets are not implemented directly by this class, but by the derived class TopologicalManifold (an open subset of a topological manifold being itself a topological manifold).

INPUT:

- manifold – topological manifold on which the subset is defined
- name – string; name (symbol) given to the subset
- latex_name – (default: None) string; LaTeX symbol to denote the subset; if none are provided, it is set to name

- `category` – (default: `None`) to specify the categeory; if `None`, the category for generic subsets is used

EXAMPLES:

A subset of a manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: from sage.manifolds.subset import ManifoldSubset
sage: A = ManifoldSubset(M, 'A', latex_name=r'\mathcal{A}')
sage: A
Subset A of the 2-dimensional topological manifold M
sage: latex(A)
\mathcal{A}
sage: A.is_subset(M)
True
```

Instead of importing `ManifoldSubset` in the global namespace, it is recommended to use the method `subset()` to create a new subset:

```
sage: B = M.subset('B', latex_name=r'\mathcal{B}'); B
Subset B of the 2-dimensional topological manifold M
sage: M.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset B of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M]
```

The manifold is itself a subset:

```
sage: isinstance(M, ManifoldSubset)
True
sage: M in M.subsets()
True
```

Instances of `ManifoldSubset` are parents:

```
sage: isinstance(A, Parent)
True
sage: A.category()
Category of subobjects of sets
sage: p = A.an_element(); p
Point on the 2-dimensional topological manifold M
sage: p.parent()
Subset A of the 2-dimensional topological manifold M
sage: p in A
True
sage: p in M
True
```

Element
alias of `ManifoldPoint`

ambient()
Return the ambient manifold of `self`.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: A.manifold()
2-dimensional topological manifold M
sage: A.manifold() is M
True
```

```
sage: B = A.subset('B')
sage: B.manifold() is M
True
```

An alias is `ambient`:

```
sage: A.ambient() is A.manifold()
True
```

declare_union(*dom1*, *dom2*)

Declare that the current subset is the union of two subsets.

Suppose U is the current subset, then this method declares that U

$$U = U_1 \cup U_2,$$

where $U_1 \subset U$ and $U_2 \subset U$.

INPUT:

- `dom1` – the subset U_1
- `dom2` – the subset U_2

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: B = M.subset('B')
sage: M.declare_union(A, B)
sage: A.union(B)
2-dimensional topological manifold M
```

get_subset(*name*)

Get a subset by its name.

The subset must have been previously created by the method `subset()` (or `open_subset()`)

INPUT:

- `name` – (string) name of the subset

OUTPUT:

- instance of `ManifoldSubset` (or of the derived class `TopologicalManifold` for an open subset) representing the subset whose name is `name`

EXAMPLES:

```
sage: M = Manifold(4, 'M', structure='topological')
sage: A = M.subset('A')
sage: B = A.subset('B')
sage: U = M.open_subset('U')
sage: M.list_of_subsets()
[Subset A of the 4-dimensional topological manifold M,
 Subset B of the 4-dimensional topological manifold M,
 4-dimensional topological manifold M,
 Open subset U of the 4-dimensional topological manifold M]
sage: M.get_subset('A')
Subset A of the 4-dimensional topological manifold M
sage: M.get_subset('A') is A
True
sage: M.get_subset('B') is B
```

```
True
sage: A.get_subset('B') is B
True
sage: M.get_subset('U')
Open subset U of the 4-dimensional topological manifold M
sage: M.get_subset('U') is U
True
```

intersection(*other*, *name=None*, *latex_name=None*)

Return the intersection of the current subset with another subset.

INPUT:

- `other` – another subset of the same manifold
- `name` – (default: `None`) name given to the intersection in the case the latter has to be created; the default is `self._name` inter `other._name`
- `latex_name` – (default: `None`) LaTeX symbol to denote the intersection in the case the latter has to be created; the default is built upon the symbol $\cap$

OUTPUT:

- instance of `ManifoldSubset` representing the subset that is the intersection of the current subset with `other`

EXAMPLES:

Intersection of two subsets:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = M.subset('B')
sage: c = a.intersection(b); c
Subset A_inter_B of the 2-dimensional topological manifold M
sage: a.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset A_inter_B of the 2-dimensional topological manifold M]
sage: b.list_of_subsets()
[Subset A_inter_B of the 2-dimensional topological manifold M,
 Subset B of the 2-dimensional topological manifold M]
sage: c._supersets  # random (set output)
{Subset B of the 2-dimensional topological manifold M,
 Subset A_inter_B of the 2-dimensional topological manifold M,
 Subset A of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M}
```

Some checks:

```
sage: (a.intersection(b)).is_subset(a)
True
sage: (a.intersection(b)).is_subset(a)
True
sage: a.intersection(b) is b.intersection(a)
True
sage: a.intersection(a.intersection(b)) is a.intersection(b)
True
sage: (a.intersection(b)).intersection(a) is a.intersection(b)
True
sage: M.intersection(a) is a
True
```

```
sage: a.intersection(M) is a
True
```

is_open()

Return if `self` is an open set.

This method always returns `False`, since open subsets must be constructed as instances of the subclass `TopologicalManifold` (which redefines `is_open`)

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: A.is_open()
False
```

is_subset(*other*)

Return `True` if and only if `self` is included in `other`.

EXAMPLES:

Subsets on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = a.subset('B')
sage: c = M.subset('C')
sage: a.is_subset(M)
True
sage: b.is_subset(a)
True
sage: b.is_subset(M)
True
sage: a.is_subset(b)
False
sage: c.is_subset(a)
False
```

lift(*p*)

Return the lift of `p` to the ambient manifold of `self`.

INPUT:

- `p` – point of the subset

OUTPUT:

- the same point, considered as a point of the ambient manifold

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: A = M.open_subset('A', coord_def={X: x>0})
sage: p = A((1, -2)); p
Point on the 2-dimensional topological manifold M
sage: p.parent()
Open subset A of the 2-dimensional topological manifold M
sage: q = A.lift(p); q
Point on the 2-dimensional topological manifold M
sage: q.parent()
2-dimensional topological manifold M
```

```
sage: q.coord()
(1, -2)
sage: (p == q) and (q == p)
True
```

list_of_subsets()
Return the list of subsets that have been defined on the current subset.

The list is sorted by the alphabetical names of the subsets.

OUTPUT:

- a list containing all the subsets that have been defined on the current subset

---

Note: To get the subsets as a Python set, used the method `subsets()` instead.

EXAMPLES:

Subsets of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = M.subset('V')
sage: M.list_of_subsets()
[2-dimensional topological manifold M,
 Open subset U of the 2-dimensional topological manifold M,
 Subset V of the 2-dimensional topological manifold M]
```

The method `subsets()` returns a set instead of a list:

```
sage: M.subsets()  # random (set output)
{Subset V of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M,
 Open subset U of the 2-dimensional topological manifold M}
```

manifold()
Return the ambient manifold of `self`.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: A = M.subset('A')
sage: A.manifold()
2-dimensional topological manifold M
sage: A.manifold() is M
True
sage: B = A.subset('B')
sage: B.manifold() is M
True
```

An alias is `ambient`:

```
sage: A.ambient() is A.manifold()
True
```

open_covers()
Return the list of open covers of the current subset.

If the current subset, A say, is a subset of the manifold M, an *open cover* of A is list (indexed set) $(U_i)_{i\in I}$

of open subsets of M such that

$$A \subset \bigcup_{i \in I} U_i.$$

If A is open, we ask that the above inclusion is actually an identity:

$$A = \bigcup_{i \in I} U_i.$$

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: M.open_covers()
[[2-dimensional topological manifold M]]
sage: U = M.open_subset('U')
sage: U.open_covers()
[[Open subset U of the 2-dimensional topological manifold M]]
sage: A = U.open_subset('A')
sage: B = U.open_subset('B')
sage: U.declare_union(A,B)
sage: U.open_covers()
[[Open subset U of the 2-dimensional topological manifold M],
 [Open subset A of the 2-dimensional topological manifold M,
  Open subset B of the 2-dimensional topological manifold M]]
sage: V = M.open_subset('V')
sage: M.declare_union(U,V)
sage: M.open_covers()
[[2-dimensional topological manifold M],
 [Open subset U of the 2-dimensional topological manifold M,
  Open subset V of the 2-dimensional topological manifold M],
 [Open subset A of the 2-dimensional topological manifold M,
  Open subset B of the 2-dimensional topological manifold M,
  Open subset V of the 2-dimensional topological manifold M]]
```

point (*coords=None*, *chart=None*, *name=None*, *latex_name=None*)

Define a point in `self`.

See `ManifoldPoint` for a complete documentation.

INPUT:

- `coords` – the point coordinates (as a tuple or a list) in the chart specified by `chart`
- `chart` – (default: `None`) chart in which the point coordinates are given; if `None`, the coordinates are assumed to refer to the default chart of the current subset
- `name` – (default: `None`) name given to the point
- `latex_name` – (default: `None`) LaTeX symbol to denote the point; if `None`, the LaTeX symbol is set to `name`

OUTPUT:

- the declared point, as an instance of `ManifoldPoint`

EXAMPLES:

Points on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: p = M.point((1,2), name='p'); p
```

```
Point p on the 2-dimensional topological manifold M
sage: p in M
True
sage: a = M.open_subset('A')
sage: c_uv.<u,v> = a.chart()
sage: q = a.point((-1,0), name='q'); q
Point q on the 2-dimensional topological manifold M
sage: q in a
True
sage: p._coordinates
{Chart (M, (x, y)): (1, 2)}
sage: q._coordinates
{Chart (A, (u, v)): (-1, 0)}
```

retract(*p*)

Return the retract of `p` to `self`.

INPUT:

- `p` – point of the ambient manifold

OUTPUT:

- the same point, considered as a point of the subset

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: A = M.open_subset('A', coord_def={X: x>0})
sage: p = M((1, -2)); p
Point on the 2-dimensional topological manifold M
sage: p.parent()
2-dimensional topological manifold M
sage: q = A.retract(p); q
Point on the 2-dimensional topological manifold M
sage: q.parent()
Open subset A of the 2-dimensional topological manifold M
sage: q.coord()
(1, -2)
sage: (q == p) and (p == q)
True
```

Of course, if the point does not belong to `A`, the `retract` method fails:

```
sage: p = M((-1, 3))  #  x < 0, so that p is not in A
sage: q = A.retract(p)
Traceback (most recent call last):
...
ValueError: the Point on the 2-dimensional topological manifold M
 is not in Open subset A of the 2-dimensional topological manifold M
```

subset(*name*, *latex_name=None*, *is_open=False*)

Create a subset of the current subset.

INPUT:

- `name` – name given to the subset
- `latex_name` – (default: `None`) LaTeX symbol to denote the subset; if none are provided, it is set to `name`

- is_open – (default: False) if True, the created subset is assumed to be open with respect to the manifold's topology

OUTPUT:

- the subset, as an instance of ManifoldSubset, or of the derived class TopologicalManifold if is_open is True

EXAMPLES:

Creating a subset of a manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A'); a
Subset A of the 2-dimensional topological manifold M
```

Creating a subset of A:

```
sage: b = a.subset('B', latex_name=r'\mathcal{B}'); b
Subset B of the 2-dimensional topological manifold M
sage: latex(b)
\mathcal{B}
```

We have then:

```
sage: b.is_subset(a)
True
sage: b in a.subsets()
True
```

subsets()

Return the set of subsets that have been defined on the current subset.

OUTPUT:

- a Python set containing all the subsets that have been defined on the current subset

---

Note: To get the subsets as a list, used the method list_of_subsets() instead.

EXAMPLES:

Subsets of a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: V = M.subset('V')
sage: M.subsets()  # random (set output)
{Subset V of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M,
 Open subset U of the 2-dimensional topological manifold M}
sage: type(M.subsets())
<type 'frozenset'>
sage: U in M.subsets()
True
```

The method list_of_subsets() returns a list (sorted alphabetically by the subset names) instead of a set:

```
sage: M.list_of_subsets()
[2-dimensional topological manifold M,
 Open subset U of the 2-dimensional topological manifold M,
 Subset V of the 2-dimensional topological manifold M]
```

superset (*name*, *latex_name=None*, *is_open=False*)
Create a superset of the current subset.

A *superset* is a manifold subset in which the current subset is included.

INPUT:

- `name` – name given to the superset
- `latex_name` – (default: `None`) LaTeX symbol to denote the superset; if none are provided, it is set to `name`
- `is_open` – (default: `False`) if `True`, the created subset is assumed to be open with respect to the manifold's topology

OUTPUT:

- the superset, as an instance of `ManifoldSubset` or of the derived class `TopologicalManifold` if `is_open` is `True`

EXAMPLES:

Creating some superset of a given subset:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = a.superset('B'); b
Subset B of the 2-dimensional topological manifold M
sage: b.list_of_subsets()
[Subset A of the 2-dimensional topological manifold M,
 Subset B of the 2-dimensional topological manifold M]
sage: a._supersets # random (set output)
{Subset B of the 2-dimensional topological manifold M,
 Subset A of the 2-dimensional topological manifold M,
 2-dimensional topological manifold M}
```

The superset of the whole manifold is itself:

```
sage: M.superset('SM') is M
True
```

Two supersets of a given subset are a priori different:

```
sage: c = a.superset('C')
sage: c == b
False
```

union (*other*, *name=None*, *latex_name=None*)
Return the union of the current subset with another subset.

INPUT:

- `other` – another subset of the same manifold
- `name` – (default: `None`) name given to the union in the case the latter has to be created; the default is `self._name` union `other._name`
- `latex_name` – (default: `None`) LaTeX symbol to denote the union in the case the latter has to be created; the default is built upon the symbol $\cup$

OUTPUT:

- instance of `ManifoldSubset` representing the subset that is the union of the current subset with `other`

EXAMPLES:

Union of two subsets:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: a = M.subset('A')
sage: b = M.subset('B')
sage: c = a.union(b); c
Subset A_union_B of the 2-dimensional topological manifold M
sage: a._supersets  # random (set output)
set([subset 'A_union_B' of the 2-dimensional manifold 'M',
     2-dimensional manifold 'M',
     subset 'A' of the 2-dimensional manifold 'M'])
sage: b._supersets  # random (set output)
set([subset 'B' of the 2-dimensional manifold 'M',
     2-dimensional manifold 'M',
     subset 'A_union_B' of the 2-dimensional manifold 'M'])
sage: c._subsets  # random (set output)
set([subset 'A_union_B' of the 2-dimensional manifold 'M',
    subset 'A' of the 2-dimensional manifold 'M',
    subset 'B' of the 2-dimensional manifold 'M'])
```

Some checks:

```
sage: a.is_subset(a.union(b))
True
sage: b.is_subset(a.union(b))
True
sage: a.union(b) is b.union(a)
True
sage: a.union(a.union(b)) is a.union(b)
True
sage: (a.union(b)).union(a) is a.union(b)
True
sage: a.union(M) is M
True
sage: M.union(a) is M
True
```

1.3 Manifold Structures

These classes encode the structure of a manifold.

AUTHORS:

- Travis Scrimshaw (2015-11-25): Initial version

class `sage.manifolds.structure.`**`RealTopologicalStructure`**

Bases: `sage.misc.fast_methods.Singleton`

The structure of a topological manifold over $\mathbf{R}$.

`chart`

alias of `RealChart`

`subcategory`(*cat*)

Return the subcategory of `cat` corresponding to the structure of `self`.

EXAMPLES:

```
sage: from sage.manifolds.structure import RealTopologicalStructure
sage: from sage.categories.manifolds import Manifolds
sage: RealTopologicalStructure().subcategory(Manifolds(RR))
Category of manifolds over Real Field with 53 bits of precision
```

class sage.manifolds.structure.**TopologicalStructure**

Bases: sage.misc.fast_methods.Singleton

The structure of a topological manifold over a general topological field.

chart

alias of Chart

subcategory(*cat*)

Return the subcategory of cat corresponding to the structure of self.

EXAMPLES:

```
sage: from sage.manifolds.structure import TopologicalStructure
sage: from sage.categories.manifolds import Manifolds
sage: TopologicalStructure().subcategory(Manifolds(RR))
Category of manifolds over Real Field with 53 bits of precision
```

1.4 Points of Topological Manifolds

The class ManifoldPoint implements points of a topological manifold.

A ManifoldPoint object can have coordinates in various charts defined on the manifold. Two points are declared equal if they have the same coordinates in the same chart.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version

REFERENCES:

- [Lee11] J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)
- [Lee13] J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York, 2013)

EXAMPLES:

Defining a point in $\mathbf{R}^3$ by its spherical coordinates:

```
sage: M = Manifold(3, 'R^3', structure='topological')
sage: U = M.open_subset('U')  # the complement of the half-plane (y=0, x>=0)
sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
```

We construct the point in the coordinates in the default chart of U (c_spher):

```
sage: p = U((1, pi/2, pi), name='P')
sage: p
Point P on the 3-dimensional topological manifold R^3
sage: latex(p)
P
sage: p in U
True
sage: p.parent()
Open subset U of the 3-dimensional topological manifold R^3
sage: c_spher(p)
```

```
(1, 1/2*pi, pi)
sage: p.coordinates(c_spher) # equivalent to above
(1, 1/2*pi, pi)
```

Computing the coordinates of `p` in a new chart:

```
sage: c_cart.<x,y,z> = U.chart() # Cartesian coordinates on U
sage: spher_to_cart = c_spher.transition_map(c_cart,
....:                     [r*sin(th)*cos(ph), r*sin(th)*sin(ph), r*cos(th)])
sage: c_cart(p)  # evaluate P's Cartesian coordinates
(-1, 0, 0)
```

Points can be compared:

```
sage: p1 = U((1, pi/2, pi))
sage: p == p1
True
sage: q = U((1,2,3), chart=c_cart, name='Q') # point defined by its Cartesian coordinates
sage: p == q
False
```

class sage.manifolds.point.**ManifoldPoint**(*parent*, *coords=None*, *chart=None*, *name=None*, *latex_name=None*, *check_coords=True*)

Bases: sage.structure.element.Element

Point of a topological manifold.

This is a Sage *element* class, the corresponding *parent* class being TopologicalManifold or ManifoldSubset.

INPUT:

- `parent` – the manifold subset to which the point belongs
- `coords` – (default: `None`) the point coordinates (as a tuple or a list) in the chart `chart`
- `chart` – (default: `None`) chart in which the coordinates are given; if `None`, the coordinates are assumed to refer to the default chart of `parent`
- `name` – (default: `None`) name given to the point
- `latex_name` – (default: `None`) LaTeX symbol to denote the point; if `None`, the LaTeX symbol is set to `name`
- `check_coords` – (default: `True`) determines whether `coords` are valid coordinates for the chart `chart`; for symbolic coordinates, it is recommended to set `check_coords` to `False`

EXAMPLES:

A point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: (a, b) = var('a b') # generic coordinates for the point
sage: p = M.point((a, b), name='P'); p
Point P on the 2-dimensional topological manifold M
sage: p.coordinates()  # coordinates of P in the subset's default chart
(a, b)
```

Since points are Sage *elements*, the *parent* of which being the subset on which they are defined, it is equivalent to write:

```
sage: p = M((a, b), name='P'); p
Point P on the 2-dimensional topological manifold M
```

A point is an element of the manifold subset in which it has been defined:

```
sage: p in M
True
sage: p.parent()
2-dimensional topological manifold M
sage: U = M.open_subset('U', coord_def={c_xy: x>0})
sage: q = U.point((2,1), name='q')
sage: q.parent()
Open subset U of the 2-dimensional topological manifold M
sage: q in U
True
sage: q in M
True
```

By default, the LaTeX symbol of the point is deduced from its name:

```
sage: latex(p)
P
```

But it can be set to any value:

```
sage: p = M.point((a, b), name='P', latex_name=r'\mathcal{P}')
sage: latex(p)
\mathcal{P}
```

Points can be drawn in 2D or 3D graphics thanks to the method `plot()`.

add_coord(*coords*, *chart=None*)

Adds some coordinates in the specified chart.

The previous coordinates with respect to other charts are kept. To clear them, use `set_coord()` instead.

INPUT:

- `coords` – the point coordinates (as a tuple or a list)
- `chart` – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

Warning: If the point has already coordinates in other charts, it is the user's responsibility to make sure that the coordinates to be added are consistent with them.

EXAMPLES:

Setting coordinates to a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point()
```

We give the point some coordinates in the manifold's default chart:

```
sage: p.add_coordinates((2,-3))
sage: p.coordinates()
(2, -3)
sage: X(p)
(2, -3)
```

A shortcut for `add_coordinates` is `add_coord`:

```
sage: p.add_coord((2,-3))
sage: p.coord()
(2, -3)
```

Let us introduce a second chart on the manifold:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
```

If we add coordinates for `p` in chart `Y`, those in chart `X` are kept:

```
sage: p.add_coordinates((-1,5), chart=Y)
sage: p._coordinates  # random (dictionary output)
{Chart (M, (u, v)): (-1, 5), Chart (M, (x, y)): (2, -3)}
```

On the contrary, with the method `set_coordinates()`, the coordinates in charts different from `Y` would be lost:

```
sage: p.set_coordinates((-1,5), chart=Y)
sage: p._coordinates
{Chart (M, (u, v)): (-1, 5)}
```

add_coordinates (*coords*, *chart=None*)

Adds some coordinates in the specified chart.

The previous coordinates with respect to other charts are kept. To clear them, use `set_coord()` instead.

INPUT:

- `coords` – the point coordinates (as a tuple or a list)
- `chart` – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

Warning: If the point has already coordinates in other charts, it is the user's responsibility to make sure that the coordinates to be added are consistent with them.

EXAMPLES:

Setting coordinates to a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point()
```

We give the point some coordinates in the manifold's default chart:

```
sage: p.add_coordinates((2,-3))
sage: p.coordinates()
(2, -3)
sage: X(p)
(2, -3)
```

A shortcut for `add_coordinates` is `add_coord`:

```
sage: p.add_coord((2,-3))
sage: p.coord()
(2, -3)
```

Let us introduce a second chart on the manifold:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
```

If we add coordinates for `p` in chart `Y`, those in chart `X` are kept:

```
sage: p.add_coordinates((-1,5), chart=Y)
sage: p._coordinates  # random (dictionary output)
{Chart (M, (u, v)): (-1, 5), Chart (M, (x, y)): (2, -3)}
```

On the contrary, with the method `set_coordinates()`, the coordinates in charts different from `Y` would be lost:

```
sage: p.set_coordinates((-1,5), chart=Y)
sage: p._coordinates
{Chart (M, (u, v)): (-1, 5)}
```

coord(*chart=None*, *old_chart=None*)

Return the point coordinates in the specified chart.

If these coordinates are not already known, they are computed from known ones by means of change-of-chart formulas.

An equivalent way to get the coordinates of a point is to let the chart acting on the point, i.e. if `X` is a chart and `p` a point, one has `p.coordinates(chart=X) == X(p)`.

INPUT:

- `chart` – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart
- `old_chart` – (default: `None`) chart from which the coordinates in `chart` are to be computed; if `None`, a chart in which the point's coordinates are already known will be picked, privileging the subset's default chart

EXAMPLES:

Spherical coordinates of a point on $\mathbf{R}^3$:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi') # spherica
sage: p = M.point((1, pi/2, pi))
sage: p.coordinates()  # coordinates in the manifold's default chart
(1, 1/2*pi, pi)
```

Since the default chart of `M` is `c_spher`, it is equivalent to write:

```
sage: p.coordinates(c_spher)
(1, 1/2*pi, pi)
```

An alternative way to get the coordinates is to let the chart act on the point (from the very definition of a chart):

```
sage: c_spher(p)
(1, 1/2*pi, pi)
```

A shortcut for `coordinates` is `coord`:

```
sage: p.coord()
(1, 1/2*pi, pi)
```

Computing the Cartesian coordinates from the spherical ones:

```
sage: c_cart.<x,y,z> = M.chart()  # Cartesian coordinates
sage: c_spher.transition_map(c_cart, [r*sin(th)*cos(ph),
....:                                 r*sin(th)*sin(ph), r*cos(th)])
Change of coordinates from Chart (M, (r, th, ph)) to Chart (M, (x, y, z))
```

The computation is performed by means of the above change of coordinates:

```
sage: p.coord(c_cart)
(-1, 0, 0)
sage: p.coord(c_cart) == c_cart(p)
True
```

Coordinates of a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: (a, b) = var('a b') # generic coordinates for the point
sage: P = M.point((a, b), name='P')
```

Coordinates of `P` in the manifold's default chart:

```
sage: P.coord()
(a, b)
```

Coordinates of `P` in a new chart:

```
sage: c_uv.<u,v> = M.chart()
sage: ch_xy_uv = c_xy.transition_map(c_uv, [x-y, x+y])
sage: P.coord(c_uv)
(a - b, a + b)
```

Coordinates of `P` in a third chart:

```
sage: c_wz.<w,z> = M.chart()
sage: ch_uv_wz = c_uv.transition_map(c_wz, [u^3, v^3])
sage: P.coord(c_wz, old_chart=c_uv)
(a^3 - 3*a^2*b + 3*a*b^2 - b^3, a^3 + 3*a^2*b + 3*a*b^2 + b^3)
```

Actually, in the present case, it is not necessary to specify `old_chart='uv'`. Note that the first command erases all the coordinates except those in the chart `c_uv`:

```
sage: P.set_coord((a-b, a+b), c_uv)
sage: P._coordinates
{Chart (M, (u, v)): (a - b, a + b)}
sage: P.coord(c_wz)
(a^3 - 3*a^2*b + 3*a*b^2 - b^3, a^3 + 3*a^2*b + 3*a*b^2 + b^3)
sage: P._coordinates  # random (dictionary output)
{Chart (M, (u, v)): (a - b, a + b),
 Chart (M, (w, z)): (a^3 - 3*a^2*b + 3*a*b^2 - b^3,
                     a^3 + 3*a^2*b + 3*a*b^2 + b^3)}
```

coordinates(*chart=None*, *old_chart=None*)

Return the point coordinates in the specified chart.

If these coordinates are not already known, they are computed from known ones by means of change-of-chart formulas.

An equivalent way to get the coordinates of a point is to let the chart acting on the point, i.e. if `X` is a chart and `p` a point, one has `p.coordinates(chart=X) == X(p)`.

INPUT:

- `chart` – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart
- `old_chart` – (default: `None`) chart from which the coordinates in `chart` are to be computed; if `None`, a chart in which the point's coordinates are already known will be picked, privileging the subset's default chart

EXAMPLES:

Spherical coordinates of a point on $\mathbf{R}^3$:

```
sage: M = Manifold(3, 'M', structure='topological')
sage: c_spher.<r,th,ph> = M.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi') # spherica
sage: p = M.point((1, pi/2, pi))
sage: p.coordinates()  # coordinates in the manifold's default chart
(1, 1/2*pi, pi)
```

Since the default chart of `M` is `c_spher`, it is equivalent to write:

```
sage: p.coordinates(c_spher)
(1, 1/2*pi, pi)
```

An alternative way to get the coordinates is to let the chart act on the point (from the very definition of a chart):

```
sage: c_spher(p)
(1, 1/2*pi, pi)
```

A shortcut for `coordinates` is `coord`:

```
sage: p.coord()
(1, 1/2*pi, pi)
```

Computing the Cartesian coordinates from the spherical ones:

```
sage: c_cart.<x,y,z> = M.chart()  # Cartesian coordinates
sage: c_spher.transition_map(c_cart, [r*sin(th)*cos(ph),
....:                                 r*sin(th)*sin(ph), r*cos(th)])
Change of coordinates from Chart (M, (r, th, ph)) to Chart (M, (x, y, z))
```

The computation is performed by means of the above change of coordinates:

```
sage: p.coord(c_cart)
(-1, 0, 0)
sage: p.coord(c_cart) == c_cart(p)
True
```

Coordinates of a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: (a, b) = var('a b') # generic coordinates for the point
sage: P = M.point((a, b), name='P')
```

Coordinates of `P` in the manifold's default chart:

```
sage: P.coord()
(a, b)
```

Coordinates of `P` in a new chart:

```
sage: c_uv.<u,v> = M.chart()
sage: ch_xy_uv = c_xy.transition_map(c_uv, [x-y, x+y])
sage: P.coord(c_uv)
(a - b, a + b)
```

Coordinates of `P` in a third chart:

```
sage: c_wz.<w,z> = M.chart()
sage: ch_uv_wz = c_uv.transition_map(c_wz, [u^3, v^3])
sage: P.coord(c_wz, old_chart=c_uv)
(a^3 - 3*a^2*b + 3*a*b^2 - b^3, a^3 + 3*a^2*b + 3*a*b^2 + b^3)
```

Actually, in the present case, it is not necessary to specify `old_chart='uv'`. Note that the first command erases all the coordinates except those in the chart `c_uv`:

```
sage: P.set_coord((a-b, a+b), c_uv)
sage: P._coordinates
{Chart (M, (u, v)): (a - b, a + b)}
sage: P.coord(c_wz)
(a^3 - 3*a^2*b + 3*a*b^2 - b^3, a^3 + 3*a^2*b + 3*a*b^2 + b^3)
sage: P._coordinates  # random (dictionary output)
{Chart (M, (u, v)): (a - b, a + b),
 Chart (M, (w, z)): (a^3 - 3*a^2*b + 3*a*b^2 - b^3,
                     a^3 + 3*a^2*b + 3*a*b^2 + b^3)}
```

set_coord(*coords*, *chart=None*)

Sets the point coordinates in the specified chart.

Coordinates with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use the method `add_coord()` instead.

INPUT:

- `coords` – the point coordinates (as a tuple or a list)
- `chart` – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

EXAMPLES:

Setting coordinates to a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point()
```

We set the coordinates in the manifold's default chart:

```
sage: p.set_coordinates((2,-3))
sage: p.coordinates()
(2, -3)
sage: X(p)
(2, -3)
```

A shortcut for `set_coordinates` is `set_coord`:

```
sage: p.set_coord((2,-3))
sage: p.coord()
(2, -3)
```

Let us introduce a second chart on the manifold:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
```

If we set the coordinates of `p` in chart `Y`, those in chart `X` are lost:

```
sage: Y(p)
(-1, 5)
sage: p.set_coord(Y(p), chart=Y)
sage: p._coordinates
{Chart (M, (u, v)): (-1, 5)}
```

set_coordinates(*coords*, *chart=None*)

Sets the point coordinates in the specified chart.

Coordinates with respect to other charts are deleted, in order to avoid any inconsistency. To keep them, use the method `add_coord()` instead.

INPUT:

- `coords` – the point coordinates (as a tuple or a list)
- `chart` – (default: `None`) chart in which the coordinates are given; if none are provided, the coordinates are assumed to refer to the subset's default chart

EXAMPLES:

Setting coordinates to a point on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: p = M.point()
```

We set the coordinates in the manifold's default chart:

```
sage: p.set_coordinates((2,-3))
sage: p.coordinates()
(2, -3)
sage: X(p)
(2, -3)
```

A shortcut for `set_coordinates` is `set_coord`:

```
sage: p.set_coord((2,-3))
sage: p.coord()
(2, -3)
```

Let us introduce a second chart on the manifold:

```
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
```

If we set the coordinates of `p` in chart `Y`, those in chart `X` are lost:

```
sage: Y(p)
(-1, 5)
sage: p.set_coord(Y(p), chart=Y)
sage: p._coordinates
{Chart (M, (u, v)): (-1, 5)}
```

1.5 Coordinate Charts

The class `Chart` implements coordinate charts on a topological manifold over a topological field K. The subclass `RealChart` is devoted to the case $K = \mathbf{R}$, for which the concept of coordinate range is meaningful.

Transition maps between charts are implemented via the class `CoordChange`.

AUTHORS:

- Eric Gourgoulhon, Michal Bejger (2013-2015) : initial version
- Travis Scrimshaw (2015): review tweaks

REFERENCES:

- Chap. 2 of [Lee11] J.M. Lee: *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011)
- Chap. 1 of [Lee13] J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York) (2013)

class `sage.manifolds.chart.`**`Chart`**(*domain*, *coordinates=''*, *names=None*)

Bases: `sage.structure.unique_representation.UniqueRepresentation`, `sage.structure.sage_object.SageObject`

Chart on a topological manifold.

Given a topological manifold M of dimension n over a topological field K, a *chart* on M is a pair (U, φ), where U is an open subset of M and $\varphi : U \to V \subset K^n$ is a homeomorphism from U to an open subset V of K^n.

The components $(x^1, \ldots, x^n)$ of φ, defined by $\varphi(p) = (x^1(p), \ldots, x^n(p)) \in K^n$ for any point $p \in U$, are called the *coordinates* of the chart (U, φ).

INPUT:

- `domain` – open subset U on which the chart is defined (must be an instance of `TopologicalManifold`)
- `coordinates` – (default: `''` (empty string)) the string defining the coordinate symbols, see below
- `names` – (default: `None`) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator <,> is used)

The string `coordinates` has the space `' '` as a separator and each item has at most two fields, separated by a colon (`:`):

1. the coordinate symbol (a letter or a few letters);
2. (optional) the LaTeX spelling of the coordinate, if not provided the coordinate symbol given in the first field will be used.

If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of LaTeX's backslash character (see examples below). If no LaTeX spelling is to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator <,> is used via Sage preparser (see examples below).

EXAMPLES:

A chart on a complex 2-dimensional topological manifold:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X = M.chart('x y'); X
Chart (M, (x, y))
sage: latex(X)
\left(M,(x, y)\right)
sage: type(X)
<class 'sage.manifolds.chart.Chart'>
```

To manipulate the coordinates (x, y) as global variables, one has to set:

```
sage: x,y = X[:]
```

However, a shortcut is to use the declarator `<x,y>` in the left-hand side of the chart declaration (there is then no need to pass the string `'x y'` to `chart()`):

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart(); X
Chart (M, (x, y))
```

The coordinates are then immediately accessible:

```
sage: y
y
sage: x is X[0] and y is X[1]
True
```

Note that `x` and `y` declared in `<x,y>` are mere Python variable names and do not have to coincide with the coordinate symbols; for instance, one may write:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x1,y1> = M.chart('x y'); X
Chart (M, (x, y))
```

Then `y` is not known as a global Python variable and the coordinate y is accessible only through the global variable `y1`:

```
sage: y1
y
sage: latex(y1)
y
sage: y1 is X[1]
True
```

However, having the name of the Python variable coincide with the coordinate symbol is quite convenient; so it is recommended to declare:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart()
```

In the above example, the chart X covers entirely the manifold `M`:

```
sage: X.domain()
Complex 2-dimensional topological manifold M
```

Of course, one may declare a chart only on an open subset of `M`:

```
sage: U = M.open_subset('U')
sage: Y.<z1, z2> = U.chart(r'z1:\zeta_1 z2:\zeta_2'); Y
Chart (U, (z1, z2))
sage: Y.domain()
Open subset U of the Complex 2-dimensional topological manifold M
```

In the above declaration, we have also specified some LaTeX writing of the coordinates different from the text one:

```
sage: latex(z1)
{\zeta_1}
```

Note the prefix `r` in front of the string `r'z1:\zeta_1 z2:\zeta_2'`; it makes sure that the backslash character is treated as an ordinary character, to be passed to the LaTeX interpreter.

Coordinates are Sage symbolic variables (see `sage.symbolic.expression`):

```
sage: type(z1)
<type 'sage.symbolic.expression.Expression'>
```

In addition to the Python variable name provided in the operator `<.,.>`, the coordinates are accessible by their indices:

```
sage: Y[0], Y[1]
(z1, z2)
```

The index range is that declared during the creation of the manifold. By default, it starts at 0, but this can be changed via the parameter `start_index`:

```
sage: M1 = Manifold(2, 'M_1', field='complex', structure='topological',
....:               start_index=1)
sage: Z.<u,v> = M1.chart()
sage: Z[1], Z[2]
(u, v)
```

The full set of coordinates is obtained by means of the slice operator `[:]`:

```
sage: Y[:]
(z1, z2)
```

Some partial sets of coordinates:

```
sage: Y[:1]
(z1,)
sage: Y[1:]
(z2,)
```

Each constructed chart is automatically added to the manifold's user atlas:

```
sage: M.atlas()
[Chart (M, (x, y)), Chart (U, (z1, z2))]
```

and to the atlas of the chart's domain:

```
sage: U.atlas()
[Chart (U, (z1, z2))]
```

Manifold subsets have a *default chart*, which, unless changed via the method `set_default_chart()`, is the first defined chart on the subset (or on a open subset of it):

```
sage: M.default_chart()
Chart (M, (x, y))
sage: U.default_chart()
Chart (U, (z1, z2))
```

The default charts are not privileged charts on the manifold, but rather charts whose name can be skipped in the argument list of functions having an optional `chart=` argument.

The chart map φ acting on a point is obtained by passing it as an input to the map:

```
sage: p = M.point((1+i, 2), chart=X); p
Point on the Complex 2-dimensional topological manifold M
sage: X(p)
(I + 1, 2)
```

```
sage: X(p) == p.coord(X)
True
```

See also:

sage.manifolds.chart.RealChart for charts on topological manifolds over $\mathbf{R}$.

add_restrictions (*restrictions*)

Add some restrictions on the coordinates.

INPUT:

- restrictions – list of restrictions on the coordinates, in addition to the ranges declared by the intervals specified in the chart constructor

A restriction can be any symbolic equality or inequality involving the coordinates, such as x > y or x^2 + y^2 != 0. The items of the list restrictions are combined with the and operator; if some restrictions are to be combined with the or operator instead, they have to be passed as a tuple in some single item of the list restrictions. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means (x > y) and ((x != 0) or (y != 0)) and (z^2 < x). If the list restrictions contains only one item, this item can be passed as such, i.e. writing x > y instead of the single element list [x > y].

EXAMPLES:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.add_restrictions(abs(x) > 1)
sage: X.valid_coordinates(2+i, 1)
True
sage: X.valid_coordinates(i, 1)
False
```

domain()

Return the open subset on which the chart is defined.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.domain()
2-dimensional topological manifold M
sage: U = M.open_subset('U')
sage: Y.<u,v> = U.chart()
sage: Y.domain()
Open subset U of the 2-dimensional topological manifold M
```

manifold()

Return the manifold on which the chart is defined.

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: U = M.open_subset('U')
sage: X.<x,y> = U.chart()
sage: X.manifold()
2-dimensional topological manifold M
sage: X.domain()
Open subset U of the 2-dimensional topological manifold M
```

restrict(*subset*, *restrictions=None*)

Return the restriction of the chart to some open subset of its domain.

If the current chart is (U, φ), a *restriction* (or *subchart*) is a chart (V, ψ) such that $V \subset U$ and $\psi = \varphi|_V$.

If such subchart has not been defined yet, it is constructed here.

The coordinates of the subchart bare the same names as the coordinates of the current chart.

INPUT:

- `subset` – open subset V of the chart domain U (must be an instance of `TopologicalManifold`)
- `restrictions` – (default: `None`) list of coordinate restrictions defining the subset V

A restriction can be any symbolic equality or inequality involving the coordinates, such as `x > y` or `x^2 + y^2 != 0`. The items of the list `restrictions` are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list `restrictions`. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means (`x > y`) and ((`x != 0`) or (`y != 0`)) and (`z^2 < x`). If the list `restrictions` contains only one item, this item can be passed as such, i.e. writing `x > y` instead of the single element list `[x > y]`.

OUTPUT:

- chart (V, ψ), as an instance of `Chart`.

EXAMPLES:

Coordinates on the unit open ball of $\mathbf{C}^2$ as a subchart of the global coordinates of $\mathbf{C}^2$:

```
sage: M = Manifold(2, 'C^2', field='complex', structure='topological')
sage: X.<z1, z2> = M.chart()
sage: B = M.open_subset('B')
sage: X_B = X.restrict(B, abs(z1)^2 + abs(z2)^2 < 1); X_B
Chart (B, (z1, z2))
```

transition_map(*other*, *transformations*, *intersection_name=None*, *restrictions1=None*, *restrictions2=None*)

Construct the transition map between the current chart, (U, φ) say, and another one, (V, ψ) say.

If n is the manifold's dimension, the *transition map* is the map

$$\psi \circ \varphi^{-1} : \varphi(U \cap V) \subset K^n \to \psi(U \cap V) \subset K^n,$$

where K is the manifold's base field. In other words, the transition map expresses the coordinates $(y^1, \ldots, y^n)$ of (V, ψ) in terms of the coordinates $(x^1, \ldots, x^n)$ of (U, φ) on the open subset where the two charts intersect, i.e. on $U \cap V$.

INPUT:

- `other` – the chart (V, ψ)
- `transformations` – tuple (or list) $(Y_1, \ldots, Y_n)$, where Y_i is the symbolic expression of the coordinate y^i in terms of the coordinates $(x^1, \ldots, x^n)$
- `intersection_name` – (default: `None`) name to be given to the subset $U \cap V$ if the latter differs from U or V

- `restrictions1` – (default: `None`) list of conditions on the coordinates of the current chart that define $U \cap V$ if the latter differs from U
- `restrictions2` – (default: `None`) list of conditions on the coordinates of the chart (V, ψ) that define $U \cap V$ if the latter differs from V

A restriction can be any symbolic equality or inequality involving the coordinates, such as `x > y` or `x^2 + y^2 != 0`. The items of the list `restrictions` are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list `restrictions`. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means (`x > y`) and ((`x != 0`) or (`y != 0`)) and (`z^2 < x`). If the list `restrictions` contains only one item, this item can be passed as such, i.e. writing `x > y` instead of the single element list `[x > y]`.

OUTPUT:

- the transition map $\psi \circ \varphi^{-1}$ defined on $U \cap V$, as an instance of `CoordChange`

EXAMPLES:

Transition map between two stereographic charts on the circle S^1:

```
sage: M = Manifold(1, 'S^1', structure='topological')
sage: U = M.open_subset('U') # Complement of the North pole
sage: cU.<x> = U.chart() # Stereographic chart from the North pole
sage: V = M.open_subset('V') # Complement of the South pole
sage: cV.<y> = V.chart() # Stereographic chart from the South pole
sage: M.declare_union(U,V)   # S^1 is the union of U and V
sage: trans = cU.transition_map(cV, 1/x, intersection_name='W',
....:                           restrictions1= x!=0, restrictions2 = y!=0)
sage: trans
Change of coordinates from Chart (W, (x,)) to Chart (W, (y,))
sage: trans.display()
y = 1/x
```

The subset W, intersection of U and V, has been created by `transition_map()`:

```
sage: M.list_of_subsets()
[1-dimensional topological manifold S^1,
 Open subset U of the 1-dimensional topological manifold S^1,
 Open subset V of the 1-dimensional topological manifold S^1,
 Open subset W of the 1-dimensional topological manifold S^1]
sage: W = M.list_of_subsets()[3]
sage: W is U.intersection(V)
True
sage: M.atlas()
[Chart (U, (x,)), Chart (V, (y,)), Chart (W, (x,)), Chart (W, (y,))]
```

Transition map between the spherical chart and the Cartesian one on $\mathbf{R}^2$:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart()
sage: U = M.open_subset('U') # the complement of the half line {y=0, x >= 0}
sage: c_spher.<r,phi> = U.chart(r'r:(0,+oo) phi:(0,2*pi):\phi')
sage: trans = c_spher.transition_map(c_cart, (r*cos(phi), r*sin(phi)),
....:                                restrictions2=(y!=0, x<0))
sage: trans
Change of coordinates from Chart (U, (r, phi)) to Chart (U, (x, y))
sage: trans.display()
```

```
x = r*cos(phi)
y = r*sin(phi)
```

In this case, no new subset has been created since $U \cap M = U$:

```
sage: M.list_of_subsets()
[2-dimensional topological manifold R^2,
 Open subset U of the 2-dimensional topological manifold R^2]
```

but a new chart has been created: $(U, (x, y))$:

```
sage: M.atlas()
[Chart (R^2, (x, y)), Chart (U, (r, phi)), Chart (U, (x, y))]
```

valid_coordinates(**coordinates*, ***kwds*)

Check whether a tuple of coordinates can be the coordinates of a point in the chart domain.

INPUT:

- `*coordinates` – coordinate values
- `**kwds` – options:
  - `parameters=None`, dictionary to set numerical values to some parameters (see example below)

OUTPUT:

- `True` if the coordinate values are admissible in the chart image, `False` otherwise

EXAMPLES:

```
sage: M = Manifold(2, 'M', field='complex', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.add_restrictions([abs(x)<1, y!=0])
sage: X.valid_coordinates(0, i)
True
sage: X.valid_coordinates(i, 1)
False
sage: X.valid_coordinates(i/2, 1)
True
sage: X.valid_coordinates(i/2, 0)
False
sage: X.valid_coordinates(2, 0)
False
```

Example of use with the keyword `parameters` to set a specific value to a parameter appearing in the coordinate restrictions:

```
sage: var('a')  # the parameter is a symbolic variable
a
sage: Y.<u,v> = M.chart()
sage: Y.add_restrictions(abs(v)<a)
sage: Y.valid_coordinates(1, i, parameters={a: 2})  # setting a=2
True
sage: Y.valid_coordinates(1, 2*i, parameters={a: 2})
False
```

class sage.manifolds.chart.**CoordChange**(*chart1*, *chart2*, **transformations*)

Bases: sage.structure.sage_object.SageObject

Transition map between two charts of a topological manifold.

Giving two coordinate charts (U, φ) and (V, ψ) on a topological manifold M of dimension n over a topological field K, the *transition map from* (U, φ) *to* (V, ψ) is the map

$$\psi \circ \varphi^{-1} : \varphi(U \cap V) \subset K^n \to \psi(U \cap V) \subset K^n.$$

In other words, the transition map $\psi \circ \varphi^{-1}$ expresses the coordinates $(y^1, \ldots, y^n)$ of (V, ψ) in terms of the coordinates $(x^1, \ldots, x^n)$ of (U, φ) on the open subset where the two charts intersect, i.e. on $U \cap V$.

INPUT:

- `chart1` – chart (U, φ)
- `chart2` – chart (V, ψ)
- `transformations` – tuple (or list) $(Y_1, \ldots, Y_2)$, where Y_i is the symbolic expression of the coordinate y^i in terms of the coordinates $(x^1, \ldots, x^n)$

EXAMPLES:

Transition map on a 2-dimensional topological manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
sage: X_to_Y
Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))
sage: type(X_to_Y)
<class 'sage.manifolds.chart.CoordChange'>
sage: X_to_Y.display()
u = x + y
v = x - y
```

disp()

Display of the coordinate transformation.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

From spherical coordinates to Cartesian ones in the plane:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: U = M.open_subset('U') # the complement of the half line {y=0, x>= 0}
sage: c_cart.<x,y> = U.chart()
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: spher_to_cart = c_spher.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: spher_to_cart.display()
x = r*cos(ph)
y = r*sin(ph)
sage: latex(spher_to_cart.display())
\left\{\begin{array}{lcl} x & = & r \cos\left({\phi}\right) \\
 y & = & r \sin\left({\phi}\right) \end{array}\right.
```

A shortcut is `disp()`:

```
sage: spher_to_cart.disp()
x = r*cos(ph)
y = r*sin(ph)
```

display()

Display of the coordinate transformation.

The output is either text-formatted (console mode) or LaTeX-formatted (notebook mode).

EXAMPLES:

From spherical coordinates to Cartesian ones in the plane:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: U = M.open_subset('U') # the complement of the half line {y=0, x>= 0}
sage: c_cart.<x,y> = U.chart()
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: spher_to_cart = c_spher.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: spher_to_cart.display()
x = r*cos(ph)
y = r*sin(ph)
sage: latex(spher_to_cart.display())
\left\{\begin{array}{lcl} x & = & r \cos\left({\phi}\right) \\
 y & = & r \sin\left({\phi}\right) \end{array}\right.
```

A shortcut is `disp()`:

```
sage: spher_to_cart.disp()
x = r*cos(ph)
y = r*sin(ph)
```

inverse()

Compute the inverse coordinate transformation.

OUTPUT:

- an instance of `CoordChange` representing the inverse of the current coordinate transformation

EXAMPLES:

Inverse of a coordinate transformation corresponding to a $\pi/3$-rotation in the plane:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart()
sage: c_uv.<u,v> = M.chart()
sage: xy_to_uv = c_xy.transition_map(c_uv, ((x - sqrt(3)*y)/2, (sqrt(3)*x + y)/2))
sage: M.coord_changes()
{(Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))}
sage: uv_to_xy = xy_to_uv.inverse(); uv_to_xy
Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y))
sage: uv_to_xy.display()
x = 1/2*sqrt(3)*v + 1/2*u
y = -1/2*sqrt(3)*u + 1/2*v
sage: M.coord_changes()  # random (dictionary output)
{(Chart (M, (u, v)),
  Chart (M, (x, y))): Change of coordinates from Chart (M, (u, v)) to Chart (M, (x, y)),
 (Chart (M, (x, y)),
  Chart (M, (u, v))): Change of coordinates from Chart (M, (x, y)) to Chart (M, (u, v))}
```

restrict(*dom1*, *dom2=None*)

Restriction to subsets.

INPUT:

- `dom1` – open subset of the domain of `chart1`
- `dom2` – (default: `None`) open subset of the domain of `chart2`; if `None`, `dom1` is assumed

OUTPUT:

- the transition map between the charts restricted to the specified subsets

EXAMPLES:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: Y.<u,v> = M.chart()
sage: X_to_Y = X.transition_map(Y, [x+y, x-y])
sage: U = M.open_subset('U', coord_def={X: x>0, Y: u+v>0})
sage: X_to_Y_U = X_to_Y.restrict(U); X_to_Y_U
Change of coordinates from Chart (U, (x, y)) to Chart (U, (u, v))
sage: X_to_Y_U.display()
u = x + y
v = x - y
```

The result is cached:

```
sage: X_to_Y.restrict(U) is X_to_Y_U
True
```

set_inverse(**transformations*, ***kwds*)

Sets the inverse of the coordinate transformation.

This is useful when the automatic computation via `inverse()` fails.

INPUT:

- `transformations` – the inverse transformations expressed as a list of the expressions of the "old" coordinates in terms of the "new" ones
- `kwds` – keyword arguments: only `verbose=True` or `verbose=False` (default) are meaningful; it determines whether the provided transformations are checked to be indeed the inverse coordinate transformations

EXAMPLES:

From spherical coordinates to Cartesian ones in the plane:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: U = M.open_subset('U') # the complement of the half line {y=0, x>= 0}
sage: c_cart.<x,y> = U.chart()
sage: c_spher.<r,ph> = U.chart(r'r:(0,+oo) ph:(0,2*pi):\phi')
sage: spher_to_cart = c_spher.transition_map(c_cart, [r*cos(ph), r*sin(ph)])
sage: spher_to_cart.set_inverse(sqrt(x^2+y^2), atan2(y,x))
sage: spher_to_cart.inverse()
Change of coordinates from Chart (U, (x, y)) to Chart (U, (r, ph))
sage: spher_to_cart.inverse().display()
r = sqrt(x^2 + y^2)
ph = arctan2(y, x)
sage: M.coord_changes()  # random (dictionary output)
{(Chart (U, (r, ph)),
  Chart (U, (x, y))): Change of coordinates from Chart (U, (r, ph)) to Chart (U, (x, y)),
 (Chart (U, (x, y)),
  Chart (U, (r, ph))): Change of coordinates from Chart (U, (x, y)) to Chart (U, (r, ph))}
```

Introducing a wrong inverse transformation (note the `x^3` typo) is revealed by setting `verbose` to `True`:

```
sage: spher_to_cart.set_inverse(sqrt(x^3+y^2), atan2(y,x), verbose=True)
Check of the inverse coordinate transformation:
   r == sqrt(r^3*cos(ph)^3 + r^2*sin(ph)^2)
   ph == arctan2(r*sin(ph), r*cos(ph))
   x == sqrt(x^3 + y^2)*x/sqrt(x^2 + y^2)
   y == sqrt(x^3 + y^2)*y/sqrt(x^2 + y^2)
```

class sage.manifolds.chart.**RealChart**(*domain*, *coordinates=''*, *names=None*)

Bases: sage.manifolds.chart.Chart

Chart on a topological manifold over $\mathbf{R}$.

Given a topological manifold M of dimension n over $\mathbf{R}$, a *chart* on M is a pair (U, φ), where U is an open subset of M and $\varphi : U \to V \subset \mathbf{R}^n$ is a homeomorphism from U to an open subset V of $\mathbf{R}^n$.

The components $(x^1, \ldots, x^n)$ of φ, defined by $\varphi(p) = (x^1(p), \ldots, x^n(p)) \in \mathbf{R}^n$ for any point $p \in U$, are called the *coordinates* of the chart (U, φ).

INPUT:

- `domain` – open subset U on which the chart is defined
- `coordinates` – (default: `''` (empty string)) string defining the coordinate symbols and ranges, see below
- `names` – (default: `None`) unused argument, except if `coordinates` is not provided; it must then be a tuple containing the coordinate symbols (this is guaranteed if the shortcut operator <,> is used)

The string `coordinates` has the space `' '` as a separator and each item has at most three fields, separated by a colon (`:`):

1. The coordinate symbol (a letter or a few letters).
2. (optional) The interval I defining the coordinate range: if not provided, the coordinate is assumed to span all $\mathbf{R}$; otherwise I must be provided in the form `(a,b)` (or equivalently `]a,b[`). The bounds `a` and `b` can be `+/-Infinity`, `Inf`, `infinity`, `inf` or `oo`. For *singular* coordinates, non-open intervals such as `[a,b]` and `(a,b]` (or equivalently `]a,b]`) are allowed. Note that the interval declaration must not contain any whitespace.
3. (optional) The LaTeX spelling of the coordinate; if not provided the coordinate symbol given in the first field will be used.

The order of the fields 2 and 3 does not matter and each of them can be omitted. If it contains any LaTeX expression, the string `coordinates` must be declared with the prefix 'r' (for "raw") to allow for a proper treatment of LaTeX backslash characters (see examples below). If no interval range and no LaTeX spelling is to be set for any coordinate, the argument `coordinates` can be omitted when the shortcut operator <,> is used via Sage preparser (see examples below).

EXAMPLES:

Cartesian coordinates on $\mathbf{R}^3$:

```
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological',
....:              start_index=1)
sage: c_cart = M.chart('x y z'); c_cart
Chart (R^3, (x, y, z))
sage: type(c_cart)
<class 'sage.manifolds.chart.RealChart'>
```

To have the coordinates accessible as global variables, one has to set:

```
sage: (x,y,z) = c_cart[:]
```

However, a shortcut is to use the declarator <x,y,z> in the left-hand side of the chart declaration (there is then no need to pass the string `'x y z'` to `chart()`):

```
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological',
....:              start_index=1)
```

```
sage: c_cart.<x,y,z> = M.chart(); c_cart
Chart (R^3, (x, y, z))
```

The coordinates are then immediately accessible:

```
sage: y
y
sage: y is c_cart[2]
True
```

Note that `x, y, z` declared in `<x,y,z>` are mere Python variable names and do not have to coincide with the coordinate symbols; for instance, one may write:

```
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological', start_index=1)
sage: c_cart.<x1,y1,z1> = M.chart('x y z'); c_cart
Chart (R^3, (x, y, z))
```

Then `y` is not known as a global variable and the coordinate y is accessible only through the global variable `y1`:

```
sage: y1
y
sage: y1 is c_cart[2]
True
```

However, having the name of the Python variable coincide with the coordinate symbol is quite convenient; so it is recommended to declare:

```
sage: forget()  # for doctests only
sage: M = Manifold(3, 'R^3', r'\RR^3', structure='topological', start_index=1)
sage: c_cart.<x,y,z> = M.chart()
```

Spherical coordinates on the subset U of $\mathbf{R}^3$ that is the complement of the half-plane $\{y = 0, x \geq 0\}$:

```
sage: U = M.open_subset('U')
sage: c_spher.<r,th,ph> = U.chart(r'r:(0,+oo) th:(0,pi):\theta ph:(0,2*pi):\phi')
sage: c_spher
Chart (U, (r, th, ph))
```

Note the prefix 'r' for the string defining the coordinates in the arguments of `chart`.

Coordinates are Sage symbolic variables (see `sage.symbolic.expression`):

```
sage: type(th)
<type 'sage.symbolic.expression.Expression'>
sage: latex(th)
{\theta}
sage: assumptions(th)
[th is real, th > 0, th < pi]
```

Coordinate are also accessible by their indices:

```
sage: x1 = c_spher[1]; x2 = c_spher[2]; x3 = c_spher[3]
sage: print x1, x2, x3
r th ph
sage: (x1, x2, x3) == (r, th, ph)
True
```

The full set of coordinates is obtained by means of the slice `[:]`:

```
sage: c_cart[:]
(x, y, z)
```

```
sage: c_spher[:]
(r, th, ph)
```

Let us check that the declared coordinate ranges have been taken into account:

```
sage: c_cart.coord_range()
x: (-oo, +oo); y: (-oo, +oo); z: (-oo, +oo)
sage: c_spher.coord_range()
r: (0, +oo); th: (0, pi); ph: (0, 2*pi)
sage: bool(th>0 and th<pi)
True
sage: assumptions()  # list all current symbolic assumptions
[x is real, y is real, z is real, r is real, r > 0, th is real,
 th > 0, th < pi, ph is real, ph > 0, ph < 2*pi]
```

The coordinate ranges are used for simplifications:

```
sage: simplify(abs(r)) # r has been declared to lie in the interval (0,+oo)
r
sage: simplify(abs(x)) # no positive range has been declared for x
abs(x)
```

Each constructed chart is automatically added to the manifold's user atlas:

```
sage: M.atlas()
[Chart (R^3, (x, y, z)), Chart (U, (r, th, ph))]
```

and to the atlas of its domain:

```
sage: U.atlas()
[Chart (U, (r, th, ph))]
```

Manifold subsets have a *default chart*, which, unless changed via the method `set_default_chart()`, is the first defined chart on the subset (or on a open subset of it):

```
sage: M.default_chart()
Chart (R^3, (x, y, z))
sage: U.default_chart()
Chart (U, (r, th, ph))
```

The default charts are not privileged charts on the manifold, but rather charts whose name can be skipped in the argument list of functions having an optional `chart=` argument.

The chart map φ acting on a point is obtained by means of the call operator, i.e. the operator `()`:

```
sage: p = M.point((1,0,-2)); p
Point on the 3-dimensional topological manifold R^3
sage: c_cart(p)
(1, 0, -2)
sage: c_cart(p) == p.coord(c_cart)
True
sage: q = M.point((2,pi/2,pi/3), chart=c_spher) # point defined by its spherical coordinates
sage: c_spher(q)
(2, 1/2*pi, 1/3*pi)
sage: c_spher(q) == q.coord(c_spher)
True
sage: a = U.point((1,pi/2,pi)) # the default coordinates on U are the spherical ones
sage: c_spher(a)
(1, 1/2*pi, pi)
sage: c_spher(a) == a.coord(c_spher)
```

```
True
```

Cartesian coordinates on U as an example of chart construction with coordinate restrictions: since U is the complement of the half-plane $\{y = 0, x \geq 0\}$, we must have $y \neq 0$ or $x < 0$ on U. Accordingly, we set:

```
sage: c_cartU.<x,y,z> = U.chart()
sage: c_cartU.add_restrictions((y!=0, x<0))
sage: U.atlas()
[Chart (U, (r, th, ph)), Chart (U, (x, y, z))]
sage: M.atlas()
[Chart (R^3, (x, y, z)), Chart (U, (r, th, ph)), Chart (U, (x, y, z))]
sage: c_cartU.valid_coordinates(-1,0,2)
True
sage: c_cartU.valid_coordinates(1,0,2)
False
sage: c_cart.valid_coordinates(1,0,2)
True
```

Note that, as an example, the following would have meant $y \neq 0$ *and* $x < 0$:

```
c_cartU.add_restrictions([y!=0, x<0])
```

add_restrictions (*restrictions*)

Add some restrictions on the coordinates.

INPUT:

- `restrictions` – list of restrictions on the coordinates, in addition to the ranges declared by the intervals specified in the chart constructor

A restriction can be any symbolic equality or inequality involving the coordinates, such as `x > y` or `x^2 + y^2 != 0`. The items of the list `restrictions` are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list `restrictions`. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means `(x > y)` and `((x != 0) or (y != 0))` and `(z^2 < x)`. If the list `restrictions` contains only one item, this item can be passed as such, i.e. writing `x > y` instead of the single element list `[x > y]`.

EXAMPLES:

Cartesian coordinates on the open unit disc in $\mathbf{R}^2$:

```
sage: M = Manifold(2, 'M', structure='topological') # the open unit disc
sage: X.<x,y> = M.chart()
sage: X.add_restrictions(x^2+y^2<1)
sage: X.valid_coordinates(0,2)
False
sage: X.valid_coordinates(0,1/3)
True
```

The restrictions are transmitted to subcharts:

```
sage: A = M.open_subset('A') # annulus 1/2 < r < 1
sage: X_A = X.restrict(A, x^2+y^2 > 1/4)
sage: X_A._restrictions
[x^2 + y^2 < 1, x^2 + y^2 > (1/4)]
sage: X_A.valid_coordinates(0,1/3)
False
```

```
sage: X_A.valid_coordinates(2/3,1/3)
True
```

If appropriate, the restrictions are transformed into bounds on the coordinate ranges:

```
sage: U = M.open_subset('U')
sage: X_U = X.restrict(U)
sage: X_U.coord_range()
x: (-oo, +oo); y: (-oo, +oo)
sage: X_U.add_restrictions([x<0, y>1/2])
sage: X_U.coord_range()
x: (-oo, 0); y: (1/2, +oo)
```

coord_bounds(*i=None*)

Return the lower and upper bounds of the range of a coordinate.

For a nicely formatted output, use `coord_range()` instead.

INPUT:

- `i` – (default: `None`) index of the coordinate; if `None`, the bounds of all the coordinates are returned

OUTPUT:

- the coordinate bounds as the tuple `((xmin, min_included), (xmax, max_included))` where
  - `xmin` is the coordinate lower bound
  - `min_included` is a boolean, indicating whether the coordinate can take the value `xmin`, i.e. `xmin` is a strict lower bound iff `min_included` is `False`
  - `xmin` is the coordinate upper bound
  - `max_included` is a boolean, indicating whether the coordinate can take the value `xmax`, i.e. `xmax` is a strict upper bound iff `max_included` is `False`

EXAMPLES:

Some coordinate bounds on a 2-dimensional manifold:

```
sage: forget()  # for doctests only
sage: M = Manifold(2, 'M', structure='topological')
sage: c_xy.<x,y> = M.chart('x y:[0,1)')
sage: c_xy.coord_bounds(0)  # x in (-oo,+oo) (the default)
((-Infinity, False), (+Infinity, False))
sage: c_xy.coord_bounds(1)  # y in [0,1)
((0, True), (1, False))
sage: c_xy.coord_bounds()
(((-Infinity, False), (+Infinity, False)), ((0, True), (1, False)))
sage: c_xy.coord_bounds() == (c_xy.coord_bounds(0), c_xy.coord_bounds(1))
True
```

The coordinate bounds can also be recovered via the method `coord_range()`:

```
sage: c_xy.coord_range()
x: (-oo, +oo); y: [0, 1)
sage: c_xy.coord_range(y)
y: [0, 1)
```

or via Sage's function `sage.symbolic.assumptions.assumptions()`:

```
sage: assumptions(x)
[x is real]
sage: assumptions(y)
[y is real, y >= 0, y < 1]
```

coord_range (*xx=None*)

Display the range of a coordinate (or all coordinates), as an interval.

INPUT:

- `xx` – (default: `None`) symbolic expression corresponding to a coordinate of the current chart; if `None`, the ranges of all coordinates are displayed

EXAMPLES:

Ranges of coordinates on a 2-dimensional manifold:

```
sage: M = Manifold(2, 'M', structure='topological')
sage: X.<x,y> = M.chart()
sage: X.coord_range()
x: (-oo, +oo); y: (-oo, +oo)
sage: X.coord_range(x)
x: (-oo, +oo)
sage: U = M.open_subset('U', coord_def={X: [x>1, y<pi]})
sage: XU = X.restrict(U)  # restriction of chart X to U
sage: XU.coord_range()
x: (1, +oo); y: (-oo, pi)
sage: XU.coord_range(x)
x: (1, +oo)
sage: XU.coord_range(y)
y: (-oo, pi)
```

The output is LaTeX-formatted for the notebook:

```
sage: latex(XU.coord_range(y))
y :\ \left( -\infty, \pi \right)
```

restrict (*subset*, *restrictions=None*)

Return the restriction of the chart to some open subset of its domain.

If the current chart is (U, φ), a *restriction* (or *subchart*) is a chart (V, ψ) such that $V \subset U$ and $\psi = \varphi|_V$.

If such subchart has not been defined yet, it is constructed here.

The coordinates of the subchart bare the same names as the coordinates of the current chart.

INPUT:

- `subset` – open subset V of the chart domain U (must be an instance of `TopologicalManifold`)
- `restrictions` – (default: `None`) list of coordinate restrictions defining the subset V

A restriction can be any symbolic equality or inequality involving the coordinates, such as `x > y` or `x^2 + y^2 != 0`. The items of the list `restrictions` are combined with the `and` operator; if some restrictions are to be combined with the `or` operator instead, they have to be passed as a tuple in some single item of the list `restrictions`. For example:

```
restrictions = [x > y, (x != 0, y != 0), z^2 < x]
```

means (`x > y`) and ((`x != 0`) or (`y != 0`)) and (`z^2 < x`). If the list `restrictions` contains only one item, this item can be passed as such, i.e. writing `x > y` instead of the single element list `[x > y]`.

OUTPUT:

- chart (V, ψ), as an instance of `RealChart`.

EXAMPLES:

Cartesian coordinates on the unit open disc in $\mathbf{R}^2$ as a subchart of the global Cartesian coordinates:

```
sage: M = Manifold(2, 'R^2', structure='topological')
sage: c_cart.<x,y> = M.chart() # Cartesian coordinates on R^2
sage: D = M.open_subset('D') # the unit open disc
sage: c_cart_D = c_cart.restrict(D, x^2+y^2<1)
sage: p = M.point((1/2, 0))
sage: p in D
True
sage: q = M.point((1, 2))
sage: q in D
False
```

Cartesian coordinates on the annulus $1 < \sqrt{x^2 + y^2} < 2$:

```
sage: A = M.open_subset('A')
sage: c_cart_A = c_cart.restrict(A, [x^2+y^2>1, x^2+y^2<4])
sage: p in A, q in A
(False, False)
sage: a = M.point((3/2,0))
sage: a in A
True
```

valid_coordinates(**coordinates*, ***kwds*)

Check whether a tuple of coordinates can be the coordinates of a point in the chart domain.

INPUT:

- `*coordinates` – coordinate values
- `**kwds` – options:
  - `tolerance=0`, to set the absolute tolerance in the test of coordinate ranges
  - `parameters=None`, to set some numerical values to parameters

OUTPUT:

- `True` if the coordinate values are admissible in the chart range and `False` otherwise

EXAMPLES:

Cartesian coordinates on a square interior:

```
sage: forget()  # for doctest only
sage: M = Manifold(2, 'M', structure='topological')  # the square interior
sage: X.<x,y> = M.chart('x:(-2,2) y:(-2,2)')
sage: X.valid_coordinates(0,1)
True
sage: X.valid_coordinates(-3/2,5/4)
True
sage: X.valid_coordinates(0,3)
False
```

The unit open disk inside the square:

```
sage: D = M.open_subset('D', coord_def={X: x^2+y^2<1})
sage: XD = X.restrict(D)
sage: XD.valid_coordinates(0,1)
```

```
False
sage: XD.valid_coordinates(-3/2,5/4)
False
sage: XD.valid_coordinates(-1/2,1/2)
True
sage: XD.valid_coordinates(0,0)
True
```

CHAPTER

TWO

INDICES AND TABLES

- Index
- Module Index
- Search Page

[Lee11] J.M. Lee : *Introduction to Topological Manifolds*, 2nd ed., Springer (New York) (2011).

[Lee13] J.M. Lee : *Introduction to Smooth Manifolds*, 2nd ed., Springer (New York) (2013)

[KN63] S. Kobayashi & K. Nomizu : *Foundations of Differential Geometry*, vol. 1, Interscience Publishers (New York) (1963).

[Huybrechts05] D. Huybrechts : *Complex Geometry*, Springer (Berlin) (2005).

m

sage.manifolds.chart, 42
sage.manifolds.manifold, 3
sage.manifolds.point, 33
sage.manifolds.structure, 32
sage.manifolds.subset, 21

A

add_coord() (sage.manifolds.point.ManifoldPoint method), 35
add_coordinates() (sage.manifolds.point.ManifoldPoint method), 36
add_restrictions() (sage.manifolds.chart.Chart method), 45
add_restrictions() (sage.manifolds.chart.RealChart method), 55
ambient() (sage.manifolds.subset.ManifoldSubset method), 23
atlas() (sage.manifolds.manifold.TopologicalManifold method), 12

B

base_field() (sage.manifolds.manifold.TopologicalManifold method), 12
base_field_type() (sage.manifolds.manifold.TopologicalManifold method), 12

C

Chart (class in sage.manifolds.chart), 42
chart (sage.manifolds.structure.RealTopologicalStructure attribute), 32
chart (sage.manifolds.structure.TopologicalStructure attribute), 33
chart() (sage.manifolds.manifold.TopologicalManifold method), 13
coord() (sage.manifolds.point.ManifoldPoint method), 37
coord_bounds() (sage.manifolds.chart.RealChart method), 56
coord_change() (sage.manifolds.manifold.TopologicalManifold method), 14
coord_changes() (sage.manifolds.manifold.TopologicalManifold method), 15
coord_range() (sage.manifolds.chart.RealChart method), 57
CoordChange (class in sage.manifolds.chart), 48
coordinates() (sage.manifolds.point.ManifoldPoint method), 38

D

declare_union() (sage.manifolds.subset.ManifoldSubset method), 24
default_chart() (sage.manifolds.manifold.TopologicalManifold method), 15
dim() (sage.manifolds.manifold.TopologicalManifold method), 16
dimension() (sage.manifolds.manifold.TopologicalManifold method), 16
disp() (sage.manifolds.chart.CoordChange method), 49
display() (sage.manifolds.chart.CoordChange method), 49
domain() (sage.manifolds.chart.Chart method), 45

E

Element (sage.manifolds.subset.ManifoldSubset attribute), 23

G

get_chart() (sage.manifolds.manifold.TopologicalManifold method), 16
get_subset() (sage.manifolds.subset.ManifoldSubset method), 24

I

index_generator() (sage.manifolds.manifold.TopologicalManifold method), 17
intersection() (sage.manifolds.subset.ManifoldSubset method), 25
inverse() (sage.manifolds.chart.CoordChange method), 50
irange() (sage.manifolds.manifold.TopologicalManifold method), 18
is_manifestly_coordinate_domain() (sage.manifolds.manifold.TopologicalManifold method), 18
is_open() (sage.manifolds.manifold.TopologicalManifold method), 19
is_open() (sage.manifolds.subset.ManifoldSubset method), 26
is_subset() (sage.manifolds.subset.ManifoldSubset method), 26

L

lift() (sage.manifolds.subset.ManifoldSubset method), 26
list_of_subsets() (sage.manifolds.subset.ManifoldSubset method), 27

M

Manifold() (in module sage.manifolds.manifold), 7
manifold() (sage.manifolds.chart.Chart method), 45
manifold() (sage.manifolds.subset.ManifoldSubset method), 27
ManifoldPoint (class in sage.manifolds.point), 34
ManifoldSubset (class in sage.manifolds.subset), 22

O

open_covers() (sage.manifolds.subset.ManifoldSubset method), 27
open_subset() (sage.manifolds.manifold.TopologicalManifold method), 19

P

point() (sage.manifolds.subset.ManifoldSubset method), 28

R

RealChart (class in sage.manifolds.chart), 52
RealTopologicalStructure (class in sage.manifolds.structure), 32
restrict() (sage.manifolds.chart.Chart method), 46
restrict() (sage.manifolds.chart.CoordChange method), 50
restrict() (sage.manifolds.chart.RealChart method), 57
retract() (sage.manifolds.subset.ManifoldSubset method), 29

S

sage.manifolds.chart (module), 42
sage.manifolds.manifold (module), 3
sage.manifolds.point (module), 33
sage.manifolds.structure (module), 32
sage.manifolds.subset (module), 21
set_coord() (sage.manifolds.point.ManifoldPoint method), 40
set_coordinates() (sage.manifolds.point.ManifoldPoint method), 41
set_default_chart() (sage.manifolds.manifold.TopologicalManifold method), 20

set_inverse() (sage.manifolds.chart.CoordChange method), 51
start_index() (sage.manifolds.manifold.TopologicalManifold method), 20
subcategory() (sage.manifolds.structure.RealTopologicalStructure method), 32
subcategory() (sage.manifolds.structure.TopologicalStructure method), 33
subset() (sage.manifolds.subset.ManifoldSubset method), 29
subsets() (sage.manifolds.subset.ManifoldSubset method), 30
superset() (sage.manifolds.subset.ManifoldSubset method), 30

T

top_charts() (sage.manifolds.manifold.TopologicalManifold method), 21
TopologicalManifold (class in sage.manifolds.manifold), 9
TopologicalStructure (class in sage.manifolds.structure), 33
transition_map() (sage.manifolds.chart.Chart method), 46

U

union() (sage.manifolds.subset.ManifoldSubset method), 31

V

valid_coordinates() (sage.manifolds.chart.Chart method), 48
valid_coordinates() (sage.manifolds.chart.RealChart method), 58