xAct`xCoba`

Intro

`xCoba`, a companion package to `xTensor`, provides several tools for working with bases and components. It allows the user to define bases on one or more vector bundles and to handle basis vectors, using basis indices notation. It performs component calculations such as expanding a tensor in a specified basis, changing the basis of an expression or tracing the contraction of basis dummies. `xCoba` stores and handles component values efficiently, making full use of tensor symmetries. The package knows how to express derivatives and brackets of basis vectors in terms of Christoffel and torsion tensors and how to assign values to the components of a tensor. Support for charts (coordinate fields, restriction of a field to a point, etc.) is limited.

`xCoba` is built on the twin package `xTensor`, which is loaded automatically.

Load the package

This loads the package from the default directory, for example $Home/.*Mathematica*/AddOns/Applications/xAct/ for a single–user installation under Linux. `xTensor` and `xPerm` are automatically loaded.

```
In[1]:= MemoryInUse[]

Out[1]= 3059504
```

©2003–2004 José M. Martín–García

```
In[2]:= <<xAct`xCoba`
```

```
------------------------------------------------------------------------
  --
Package xAct`xCore`  version 0.5.0, {2008, 5, 16}

CopyRight (C) 2007-2008, Jose M.
  Martin-Garcia, under the General Public License.

------------------------------------------------------------------------
  --
Package ExpressionManipulation`

CopyRight (C) 1999-2008, David J. M. Park and Ted Ersek

------------------------------------------------------------------------
  --
Package xAct`xPerm`  version 1.0.1, {2008, 5, 16}

CopyRight (C) 2003-2008, Jose M.
  Martin-Garcia, under the General Public License.

Connecting to external linux executable...

Connection established.

------------------------------------------------------------------------
  --
Package xAct`xTensor`  version 0.9.5, {2008, 5, 16}

CopyRight (C) 2002-2008, Jose M.
  Martin-Garcia, under the General Public License.

------------------------------------------------------------------------
  --
Package xAct`xCoba`  version 0.6.3, {2008, 5, 16}

CopyRight (C) 2005-2008, David Yllanes and
  Jose M. Martin-Garcia, under the General Public License.

------------------------------------------------------------------------
  --
These packages come with ABSOLUTELY NO WARRANTY; for details type
  Disclaimer[]. This is free software, and you are welcome to redistribute
  it under certain conditions. See the General Public License for details.

------------------------------------------------------------------------
  --
```

Comparing, we see that the packages take about 10Mb in *Mathematica* 5.2:

```
In[3]:= MemoryInUse[]

Out[3]= 16358680

In[4]:= Out[3] - Out[1]

Out[4]= 13299176
```

©2003–2004 José M. Martín–García

There are six contexts: `xAct`xCoba`, xAct`xTensor`, xAct`xPerm` and xAct`ExpressionManipulation` contain the respective reserved words. `System`` contains *Mathematica*'s reserved words. The current context `Global`` will contain your definitions and right now it is empty.

```
In[5]:= $ContextPath

Out[5]= {xAct`xCoba`, xAct`xTensor`, xAct`xPerm`,
         xAct`xCore`, xAct`ExpressionManipulation`, Global`, System`}

In[6]:= Context[]

Out[6]= Global`

In[7]:= ?Global`*

        Information::nomatch : No symbol matching Global`* found. More...
```

We turn off the annoying spell messages:

```
In[8]:= Off[General::spell]
        Off[General::spell1]
```

■ 0. xCoba‘ at a glance

`xCoba`` has been built on `xTensor`` and shares its approach and priorities. In particular a great deal of effort has been made to allow full compatibility between abstract and component computations. A crucial ingredient has been the formalism of *marked indices* and particularly the treatment of frame dependent objects. Perhaps the most conspicuous example is the covariant parallel derivative, which follows the concept of ordinary derivative as described on Wald's *General Relativity*. This reference has been the main inspiration for much of the notation, and the user is advised to keep it in mind when dealing with some of the non standard ideas employed by this package.

A session with `xCoba`` begins just like one with `xTensor``, defining one or more manifolds and vector bundles and the objects living on them:

Define a 3d manifold `M3`:

```
In[10]:= DefManifold[M3, 3, {a, b, c, d, e, f}]

          ** DefManifold: Defining manifold M3.

          ** DefVBundle: Defining vbundle TangentM3.
```

Define a complex vector bundle

```
In[11]:= DefVBundle[InnerC, M3, 2, {A, B, C, D, F, G}, Dagger → Complex]

          ** DefVBundle: Defining vbundle InnerC.

       ValidateSymbol::capital : System name C is overloaded as an abstract index.

       ValidateSymbol::capital : System name D is overloaded as an abstract index.

          ** DefVBundle: Defining conjugated vbundle InnerC†.
           Assuming fixed anti-isomorphism between InnerC and InnerC†
```

©2003–2004 José M. Martín–García

Define a contravariant vector v and a covariant symmetric tensor T:

```
In[12]:= DefTensor[v[a], M3]
         DefTensor[T[-a, -b], M3, Antisymmetric[{-a, -b}]]

           ** DefTensor: Defining tensor v[a].

           ** DefTensor: Defining tensor T[-a, -b].
```

Define a Riemann symmetric tensor and a covariant vector on InnerC

```
In[14]:= DefTensor[u[-A], M3, Dagger → Complex]

           ** DefTensor: Defining tensor u[-A].

           ** DefTensor: Defining tensor u†[-A†].

In[15]:= DefTensor[R[-A, -B, -C, -D], M3, RiemannSymmetric[{1, 2, 3, 4}], Dagger → Complex]

           ** DefTensor: Defining tensor R[-A, -B, -C, -D].

           ** DefTensor: Defining tensor R†[-A†, -B†, -C†, -D†].
```

Remember that, in xTensor`, we use abstract indices

```
In[16]:= T[-a, -b] v[b]

Out[16]= T_ab v^b
```

Now xCoba` lets us define bases. We have to specify its cnumbers (integers identifying the vectors of the basis) and vector bundle. Several related objects are also defined (parallel derivative and associated tensors).

```
In[17]:= DefBasis[polar, TangentM3, {0, 1, 2}]

           ** DefCovD: Defining parallel derivative PDpolar[-a].

           ** DefTensor: Defining torsion tensor TorsionPDpolar[a, -b, -c].

           ** DefTensor: Defining
            non-symmetric Christoffel tensor ChristoffelPDpolar[a, -b, -c].

           ** DefTensor: Defining vanishing Riemann tensor RiemannPDpolar[-a, -b, -c, d].

           ** DefTensor: Defining vanishing Ricci tensor RicciPDpolar[-a, -b].

           ** DefTensor: Defining antisymmetric +1 density etaUppolar[a, b, c].

           ** DefTensor: Defining antisymmetric -1 density etaDownpolar[-a, -b, -c].
```

©2003–2004 José M. Martín–García

Each basis has a colour, used to identify its associated objects in StandardForm

```
In[18]:= DefBasis[cartesian, TangentM3, {0, 1, 2}, BasisColor → RGBColor[0, 0, 1]]

** DefCovD: Defining parallel derivative PDcartesian[-a].

** DefTensor: Defining torsion tensor TorsionPDcartesian[a, -b, -c].

** DefTensor: Defining non-symmetric Christoffel tensor
 ChristoffelPDcartesian[a, -b, -c].

** DefTensor: Defining vanishing Riemann tensor
 RiemannPDcartesian[-a, -b, -c, d].

** DefTensor: Defining vanishing Ricci tensor RicciPDcartesian[-a, -b].

** DefTensor: Defining antisymmetric +1 density etaUpcartesian[a, b, c].

** DefTensor: Defining antisymmetric -1 density etaDowncartesian[-a, -b, -c].
```

We can define bases on the complex vector bundle. This generates additional objects for its parallel derivative (FRiemann and AChristoffel)

```
In[19]:= DefBasis[complex, InnerC, {4, 5}, Dagger → Complex, BasisColor → RGBColor[0, 1, 0]]

** DefCovD: Defining parallel derivative PDcomplex[-a].

** DefTensor: Defining torsion tensor TorsionPDcomplex[a, -b, -c].

** DefTensor: Defining
 non-symmetric Christoffel tensor ChristoffelPDcomplex[a, -b, -c].

** DefTensor: Defining
 vanishing Riemann tensor RiemannPDcomplex[-a, -b, -c, d].

** DefTensor: Defining vanishing Ricci tensor RicciPDcomplex[-a, -b].

** DefTensor: Defining nonsymmetric AChristoffel tensor
 AChristoffelPDcomplex[A, -b, -C].

** DefTensor: Defining nonsymmetric AChristoffel tensor
 AChristoffelPDcomplex†[A†, -b, -C†].

** DefTensor: Defining
 vanishing FRiemann tensor FRiemannPDcomplex[-a, -b, -C, D].

** DefTensor: Defining vanishing FRiemann tensor
 FRiemannPDcomplex†[-a, -b, -C†, D†].

** DefTensor: Defining antisymmetric +1 density etaUpcomplex[A, B].

** DefTensor: Defining antisymmetric +1 density etaUpcomplex†[A†, B†].

** DefTensor: Defining antisymmetric -1 density etaDowncomplex[-A, -B].

** DefTensor: Defining antisymmetric -1 density etaDowncomplex†[-A†, -B†].
```

As we advanced on the introduction, all the objects that depend on a particular basis are marked with its colour, for example here the parallel derivative of the basis

```
In[20]:= PDpolar[-a][T[-b, -c]]

Out[20]= 𝒟_a T_bc
```

©2003–2004 José M. Martín–García

Now we can use basis indices and work with specific components. A contravariant basis index (BIndex) is represented by {a, basis}, where a is a valid abstract index and basis an existing basis. A covariant basis index is represented by {-a, -basis}. If we give an integer instead of an abstract index we get a component index (CIndex):

```
In[21]:= T[{-a, -polar}, b] v[{1, cartesian}]

Out[21]= T_a^b v^1
```

Covariant component indices are represented by {i, -basis}, not by {-i, -basis}. This is because we consider the bases on the tangent and cotagent bundles to have the same cnumbers (and also allows us to use nonpositive integers as cnumbers):

```
In[22]:= v[{1, -cartesian}]

Out[22]= v_1
```

Basis vectors and transition matrices are represented by the two–index object Basis.

```
In[23]:= Basis[A, {4, -complex}]

Out[23]= e^Ā_4

In[24]:= Basis[{a, polar}, {-b, -cartesian}]

Out[24]= e^a_b
```

Bases objects are not automatically contracted with one another:

```
In[25]:= Basis[a, {-b, -polar}] Basis[-a, {c, cartesian}]

Out[25]= e_a^c e^a_b

In[26]:= Basis[a, {1, -cartesian}] T[-a, -b]

Out[26]= e^a_1 T_ab
```

We must use the function ContractBasis to force the operation.

```
In[27]:= ContractBasis[%]

Out[27]= T_1b
```

This function is quite general and allows the user to specify which indices should be acted upon.The default is total contraction:

```
In[28]:= expr = Basis[a, {1, -polar}] Basis[-c, {1, cartesian}]
            v[-a, -b, c, {d, polar}] Basis[{-d, -polar}, e]

Out[28]= e^a_1 e_c^1 e^e_d v_ab^cd

In[29]:= ContractBasis[expr]

Out[29]= v_1b^1e
```

©2003–2004 José M. Martín–García

```
In[30]:= ContractBasis[expr, polar]
```

Out[30]= $e^{a}{}_{1}\, e_{c}{}^{1}\, v_{ab}{}^{ce}$

```
In[31]:= ContractBasis[expr, a]
```

Out[31]= $e_{c}{}^{1}\, e^{e}{}_{d}\, v_{1\,b}{}^{c\,d}$

The inverse function is SeparateBasis, also quite general. It can undo the action of ContractBasis and also change the basis of the whole expression

```
In[32]:= cexpr = ContractBasis[expr]
```

Out[32]= $v_{1\,b}{}^{1\,e}$

```
In[33]:= SeparateBasis[AIndex][cexpr]
```

Out[33]= $e^{f\$290}{}_{1}\, e_{f\$291}{}^{1}\, v_{f\$290b}{}^{f\$291e}$

```
In[34]:= SeparateBasis[cartesian][cexpr]
```

Out[34]= $e_{b}{}^{f\$293}\, e^{e}{}_{f\$295}\, e^{1}{}_{f\$294}\, e_{1}{}^{f\$292}\, v_{f\$292\,f\$293}{}^{f\$294\,f\$295}$

```
In[35]:= % // ScreenDollarIndices
```

Out[35]= $e_{b}{}^{c}\, e^{e}{}_{f}\, e^{1}{}_{d}\, e_{1}{}^{a}\, v_{a\,c}{}^{d\,f}$

One of the most important features of the package is the ability to trace basis contractions,

```
In[36]:= TraceBasisDummy[v[{a, cartesian}] T[{-a, -cartesian}, -b]]
```

Out[36]= $T_{0\,b}\, v^{0} + T_{1\,b}\, v^{1} + T_{2\,b}\, v^{2}$

We can produce lists of all the basis vectors,

```
In[37]:= BasisArray[polar][a]
```

Out[37]= $\{e^{a}{}_{0},\, e^{a}{}_{1},\, e^{a}{}_{2}\}$

```
In[38]:= BasisArray[polar, cartesian, cartesian][a, b, c] // MatrixForm
```

Out[38]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} e^{a}{}_{0}\, e^{b}{}_{0}\, e^{c}{}_{0} \\ e^{a}{}_{0}\, e^{b}{}_{0}\, e^{c}{}_{1} \\ e^{a}{}_{0}\, e^{b}{}_{0}\, e^{c}{}_{2} \end{pmatrix} & \begin{pmatrix} e^{a}{}_{0}\, e^{b}{}_{1}\, e^{c}{}_{0} \\ e^{a}{}_{0}\, e^{b}{}_{1}\, e^{c}{}_{1} \\ e^{a}{}_{0}\, e^{b}{}_{1}\, e^{c}{}_{2} \end{pmatrix} & \begin{pmatrix} e^{a}{}_{0}\, e^{b}{}_{2}\, e^{c}{}_{0} \\ e^{a}{}_{0}\, e^{b}{}_{2}\, e^{c}{}_{1} \\ e^{a}{}_{0}\, e^{b}{}_{2}\, e^{c}{}_{2} \end{pmatrix} \\ \begin{pmatrix} e^{a}{}_{1}\, e^{b}{}_{0}\, e^{c}{}_{0} \\ e^{a}{}_{1}\, e^{b}{}_{0}\, e^{c}{}_{1} \\ e^{a}{}_{1}\, e^{b}{}_{0}\, e^{c}{}_{2} \end{pmatrix} & \begin{pmatrix} e^{a}{}_{1}\, e^{b}{}_{1}\, e^{c}{}_{0} \\ e^{a}{}_{1}\, e^{b}{}_{1}\, e^{c}{}_{1} \\ e^{a}{}_{1}\, e^{b}{}_{1}\, e^{c}{}_{2} \end{pmatrix} & \begin{pmatrix} e^{a}{}_{1}\, e^{b}{}_{2}\, e^{c}{}_{0} \\ e^{a}{}_{1}\, e^{b}{}_{2}\, e^{c}{}_{1} \\ e^{a}{}_{1}\, e^{b}{}_{2}\, e^{c}{}_{2} \end{pmatrix} \\ \begin{pmatrix} e^{a}{}_{2}\, e^{b}{}_{0}\, e^{c}{}_{0} \\ e^{a}{}_{2}\, e^{b}{}_{0}\, e^{c}{}_{1} \\ e^{a}{}_{2}\, e^{b}{}_{0}\, e^{c}{}_{2} \end{pmatrix} & \begin{pmatrix} e^{a}{}_{2}\, e^{b}{}_{1}\, e^{c}{}_{0} \\ e^{a}{}_{2}\, e^{b}{}_{1}\, e^{c}{}_{1} \\ e^{a}{}_{2}\, e^{b}{}_{1}\, e^{c}{}_{2} \end{pmatrix} & \begin{pmatrix} e^{a}{}_{2}\, e^{b}{}_{2}\, e^{c}{}_{0} \\ e^{a}{}_{2}\, e^{b}{}_{2}\, e^{c}{}_{1} \\ e^{a}{}_{2}\, e^{b}{}_{2}\, e^{c}{}_{2} \end{pmatrix} \end{pmatrix}$$

©2003–2004 José M. Martín–García

and lists of all the components of a tensor

```
In[39]:= ComponentArray[T[{-a, -cartesian}, {-b, -cartesian}]]

Out[39]= {{T00, T01, T02}, {T10, T11, T12}, {T20, T21, T22}}
```

Notice that T is a symmetric tensor, so this can be simplified

```
In[40]:= Simplification[%] // MatrixForm
```

Out[40]//MatrixForm=

$$\begin{pmatrix} 0 & T_{01} & T_{02} \\ -T_{01} & 0 & T_{12} \\ -T_{02} & -T_{12} & 0 \end{pmatrix}$$

We can assign rules to the components of a tensor taking the symmetries into account

```
In[41]:= values = Table[i + j, {i, 3}, {j, 3}];

In[42]:= AllComponentValues[T[{-a, -polar}, {-b, -polar}], values]

          Added dependent rule T00 → 0 for tensor T

          Added independent rule T01 → 3 for tensor T

          Added independent rule T02 → 4 for tensor T

          Added dependent rule T10 → -T01 for tensor T

          Replaced independent rule T01 → 3 by T01 → -3 for tensor T

          Added dependent rule T11 → 0 for tensor T

          Added independent rule T12 → 5 for tensor T

          Added dependent rule T20 → -T02 for tensor T

          Replaced independent rule T02 → 4 by T02 → -4 for tensor T

          Added dependent rule T21 → -T12 for tensor T

          Replaced independent rule T12 → 5 by T12 → -5 for tensor T

          Added dependent rule T22 → 0 for tensor T

Out[42]= FoldedRule[{T00 → 0, T10 → -T01, T11 → 0, T20 → -T02, T21 → -T12, T22 → 0},
          {T01 → -3, T02 → -4, T12 → -5}]

In[43]:= ColumnForm /@ TensorValues[T]

Out[43]= FoldedRule[ T00 → 0    , T01 → -3
                     T10 → -T01   T02 → -4
                     T11 → 0      T12 → -5
                     T20 → -T02
                     T21 → -T12
                     T22 → 0
```

©2003–2004 José M. Martín–García

As we can see, dependent and independent components are stored in separate sublists. We can change basis:

In[44]:=
```
$CVVerbose = False;
```

In[45]:=
```
ChangeComponents[T[-{a, cartesian}, -{b, cartesian}], T[-{a, polar}, -{b, polar}]]
```

```
Computed T_ab → e_b^c T_ac in 0.050701 Seconds

Replaced independent rule T_01 → e_0^0 T_01 - e_0^1 T_11 - e_0^2 T_12
  by T_01 → e^0_1 T_00 + e_1^1 T_01 + e_1^2 T_02 for tensor T

Replaced independent rule T_02 → e_0^0 T_02 + e_0^1 T_12 - e_0^2 T_22
  by T_02 → e^0_2 T_00 + e^1_2 T_01 + e_2^2 T_02 for tensor T

Replaced independent rule T_12 → e^0_1 T_02 + e_1^1 T_12 - e_1^2 T_22
  by T_12 → -e^0_2 T_01 + e^1_2 T_11 + e_2^2 T_12 for tensor T

Computed T_ab → e_a^c T_cb in 0.134292 Seconds
```

Out[45]=
$$\text{FoldedRule}\Big[\{T_{00}\to 0,\ T_{10}\to -T_{01},\ T_{11}\to 0,\ T_{20}\to -T_{02},\ T_{21}\to -T_{12},\ T_{22}\to 0\},\ \{T_{01}\to e^0{}_1\,T_{00}+e_1{}^1\,T_{01}+e_1{}^2\,T_{02},\ T_{02}\to e^0{}_2\,T_{00}+e^1{}_2\,T_{01}+e_2{}^2\,T_{02},\ T_{12}\to -e^0{}_2\,T_{01}+e^1{}_2\,T_{11}+e_2{}^2\,T_{12}\},\ \{T_{00}\to -T_{00},\ T_{10}\to -T_{01},\ T_{20}\to -T_{02},\ T_{11}\to -T_{11},\ T_{21}\to -T_{12},\ T_{22}\to -T_{22},\ T_{10}\to -T_{01},\ T_{20}\to -T_{02},\ T_{21}\to -T_{12}\},\ \{T_{00}\to -e_0{}^1\,T_{01}-e_0{}^2\,T_{02},\ T_{01}\to e_0{}^0\,T_{01}-e_0{}^2\,T_{12},\ T_{02}\to e_0{}^0\,T_{02}+e_0{}^1\,T_{12},\ T_{01}\to e_1{}^1\,T_{01}+e_1{}^2\,T_{02},\ T_{11}\to e^0{}_1\,T_{01}-e_1{}^2\,T_{12},\ T_{12}\to e^0{}_1\,T_{02}+e_1{}^1\,T_{12},\ T_{02}\to e^1{}_2\,T_{01}+e_2{}^2\,T_{02},\ T_{12}\to -e^0{}_2\,T_{01}+e_2{}^2\,T_{12},\ T_{22}\to e^0{}_2\,T_{02}+e^1{}_2\,T_{12}\}\Big]$$

In[46]:=
```
ColumnForm /@ TensorValues[T, {{-cartesian, -cartesian}}]
```

Out[46]=
$$\text{FoldedRule}\Big[\begin{array}{ll} T_{00}\to 0 & ,\ T_{01}\to e^0{}_1\,T_{00}+e_1{}^1\,T_{01}+e_1{}^2\,T_{02} \\ T_{10}\to -T_{01} & \ \ T_{02}\to e^0{}_2\,T_{00}+e^1{}_2\,T_{01}+e_2{}^2\,T_{02} \\ T_{11}\to 0 & \ \ T_{12}\to -e^0{}_2\,T_{01}+e^1{}_2\,T_{11}+e_2{}^2\,T_{12} \\ T_{20}\to -T_{02} & \\ T_{21}\to -T_{12} & \\ T_{22}\to 0 & \end{array}\Big]$$

Clean up:

In[47]:=
```
DeleteTensorValues[T]
```

```
Deleted values for tensor T, derivatives
 {} and bases {{-cartesian, -cartesian}}.

Deleted values for tensor T, derivatives {}
  and bases {{-cartesian, -polar}, {-polar, -cartesian}}.

Deleted values for tensor T, derivatives {} and bases {{-polar, -polar}}.
```

In[48]:=
```
$CVVerbose = True;
```

In[49]:=
```
Remove[expr, cexpr, values];
```

©2003–2004 José M. Martín–García

xCoba` introduces new definitions to work with derivatives. Remember that each basis has an associated parallel derivative (PDbasisname). We can express any derivative of a Basis object in terms of Christoffel tensors relating that derivative to the PD of the basis

```
In[50]:= PDpolar[-a][Basis[{b, polar}, -c]]

Out[50]= 0

In[51]:= PD[-a][Basis[{1, polar}, -c]]

Out[51]= Γ[𝒟]^1_ac
```

If we have a basis change a new Christoffel tensor is defined, relating the PDs of both bases

```
In[52]:= PD[-a][Basis[{c, polar}, {-b, -cartesian}]]

          ** DefTensor: Defining tensor ChristoffelPDcartesianPDpolar[a, -b, -c].

Out[52]= -Γ[𝒟, 𝒟]^c_ab

In[53]:= PD[-a][Basis[{b, cartesian}, {2, -polar}]]

Out[53]= Γ[𝒟, 𝒟]^b_a2
```

We can define charts.

```
In[54]:= DefChart[coord, M3, {0, 1, 2}, {x[], y[], z[]}]

          ** DefTensor: Defining tensor x[].

          ** DefTensor: Defining tensor y[].

          ** DefTensor: Defining tensor z[].

          ** DefCovD: Defining parallel derivative PDcoord[-a].

          ** DefTensor: Defining vanishing torsion tensor TorsionPDcoord[a, -b, -c].

          ** DefTensor: Defining
           symmetric Christoffel tensor ChristoffelPDcoord[a, -b, -c].

          ** DefTensor: Defining vanishing Riemann tensor RiemannPDcoord[-a, -b, -c, d].

          ** DefTensor: Defining vanishing Ricci tensor RicciPDcoord[-a, -b].

          ** DefTensor: Defining antisymmetric +1 density etaUpcoord[a, b, c].

          ** DefTensor: Defining antisymmetric -1 density etaDowncoord[-a, -b, -c].
```

©2003–2004 José M. Martín–García

Now the torsion tensor vanishes and the Christoffel is symmetric. Functions to restrict a field to a point or express it in terms of functions of the coordinates are currently under development.

```
In[55]:= UndefBasis /@ {polar, cartesian};

** UndefTensor: Undefined tensor ChristoffelPDcartesianPDpolar

** UndefTensor: Undefined non-symmetric Christoffel tensor ChristoffelPDpolar

** UndefTensor: Undefined vanishing Ricci tensor RicciPDpolar

** UndefTensor: Undefined vanishing Riemann tensor RiemannPDpolar

** UndefTensor: Undefined torsion tensor TorsionPDpolar

** UndefCovD: Undefined parallel derivative PDpolar

** UndefTensor: Undefined antisymmetric +1 density etaUppolar

** UndefTensor: Undefined antisymmetric -1 density etaDownpolar

** Undefined basis polar

** UndefTensor: Undefined
 non-symmetric Christoffel tensor ChristoffelPDcartesian

** UndefTensor: Undefined vanishing Ricci tensor RicciPDcartesian

** UndefTensor: Undefined vanishing Riemann tensor RiemannPDcartesian

** UndefTensor: Undefined torsion tensor TorsionPDcartesian

** UndefCovD: Undefined parallel derivative PDcartesian

** UndefTensor: Undefined antisymmetric +1 density etaUpcartesian

** UndefTensor: Undefined antisymmetric -1 density etaDowncartesian

** Undefined basis cartesian
```

©2003–2004 José M. Martín–García

Removing a complex basis also removes its complex conjugate:

```
In[56]:= UndefBasis[complex];

          ** Undefined basis complex†

          ** UndefTensor: Undefined
           nonsymmetric AChristoffel tensor  AChristoffelPDcomplex†

          ** UndefTensor: Undefined
           nonsymmetric AChristoffel tensor  AChristoffelPDcomplex

          ** UndefTensor: Undefined non-symmetric Christoffel tensor ChristoffelPDcomplex

          ** UndefTensor: Undefined vanishing FRiemann tensor FRiemannPDcomplex†

          ** UndefTensor: Undefined vanishing FRiemann tensor FRiemannPDcomplex

          ** UndefTensor: Undefined vanishing Ricci tensor RicciPDcomplex

          ** UndefTensor: Undefined vanishing Riemann tensor RiemannPDcomplex

          ** UndefTensor: Undefined torsion tensor TorsionPDcomplex

          ** UndefCovD: Undefined parallel derivative PDcomplex

          ** UndefTensor: Undefined antisymmetric +1 density etaUpcomplex†

          ** UndefTensor: Undefined antisymmetric +1 density etaUpcomplex

          ** UndefTensor: Undefined antisymmetric -1 density etaDowncomplex†

          ** UndefTensor: Undefined antisymmetric -1 density etaDowncomplex

          ** Undefined basis complex

In[57]:= UndefChart[coord]

          ** UndefTensor: Undefined tensor x

          ** UndefTensor: Undefined tensor y

          ** UndefTensor: Undefined tensor z

          ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelPDcoord

          ** UndefTensor: Undefined vanishing Ricci tensor RicciPDcoord

          ** UndefTensor: Undefined vanishing Riemann tensor RiemannPDcoord

          ** UndefTensor: Undefined vanishing torsion tensor TorsionPDcoord

          ** UndefCovD: Undefined parallel derivative PDcoord

          ** UndefTensor: Undefined antisymmetric +1 density etaUpcoord

          ** UndefTensor: Undefined antisymmetric -1 density etaDowncoord
```

©2003–2004 José M. Martín–García

1. Real and complex bases. Basis objects and basis indices

1.1. DefBasis

This section will explain how to define and work with new bases, without assuming that they have an underlying coordinate chart.

DefBasis	Define a basis
VBundleOfBasis	VBundle on which a basis lives
PDOfBasis	Parallel derivative associated to the given basis
$Bases	List of currently defined bases
BasisQ	Check existence of a given basis name

Definition of a basis.

To define a basis we only have to provide a name, a vector bundle and a list of cnumbers (whose lenght must be the dimension of the bundle). The cnumbers can include 0 and even negative integers.

```
In[58]:= DefBasis[polar, TangentM3, {0, 1, 2}]

          ** DefCovD: Defining parallel derivative PDpolar[-a].

          ** DefTensor: Defining torsion tensor TorsionPDpolar[a, -b, -c].

          ** DefTensor: Defining
           non-symmetric Christoffel tensor ChristoffelPDpolar[a, -b, -c].

          ** DefTensor: Defining vanishing Riemann tensor RiemannPDpolar[-a, -b, -c, d].

          ** DefTensor: Defining vanishing Ricci tensor RicciPDpolar[-a, -b].

          ** DefTensor: Defining antisymmetric +1 density etaUppolar[a, b, c].

          ** DefTensor: Defining antisymmetric -1 density etaDownpolar[-a, -b, -c].
```

As we can see, several other objects are automatically defined: the parallel derivative and its torsion, Christoffel, Riemann and Ricci tensors. We shall say more about them in Section 3.

```
In[59]:= PDOfBasis[polar]

Out[59]= PDpolar
```

©2003–2004 José M. Martín–García

Each basis has a colour, used to identify its associated indices and objects in StandardForm. The default for new bases is red, but we can specify a different one:

```
In[60]:= DefBasis[cartesian, TangentM3, {0, 1, 2}, BasisColor → RGBColor[0, 1, 0]]

          ** DefCovD: Defining parallel derivative PDcartesian[-a].

          ** DefTensor: Defining torsion tensor TorsionPDcartesian[a, -b, -c].

          ** DefTensor: Defining non-symmetric Christoffel tensor
           ChristoffelPDcartesian[a, -b, -c].

          ** DefTensor: Defining vanishing Riemann tensor
           RiemannPDcartesian[-a, -b, -c, d].

          ** DefTensor: Defining vanishing Ricci tensor RicciPDcartesian[-a, -b].

          ** DefTensor: Defining antisymmetric +1 density etaUpcartesian[a, b, c].

          ** DefTensor: Defining antisymmetric -1 density etaDowncartesian[-a, -b, -c].
```

We can to give different cnumbers to two bases on the same vbundle. A basis has several associated definitions and UpValues:

```
In[61]:= ?polar

          Global`polar

          BasisColor[polar] ^= RGBColor[1, 0, 0]

          BasisQ[polar] ^= True

          CNumbersOf[polar] ^= {0, 1, 2}

          Dagger[polar] ^= polar

          DependenciesOfBasis[polar] ^= {M3}

          PDOfBasis[polar] ^= PDpolar

          ServantsOf[polar] ^= {PDpolar, etaUppolar, etaDownpolar}

          VBundleOfBasis[polar] ^= TangentM3

In[62]:= $Bases

Out[62]= {polar, cartesian}
```

©2003–2004 José M. Martín–García

1.2. Basis objects and basis indices

Now that we have bases, we can use basis vectors and work with tensor componentes. Two new kinds of indices are needed:

`{a, basisname}` : Basis index (`BIndex`), `a` is an abstract index.

`{i, basisname}` : Component index (`CIndex`), `i` is a valid coordinate number.

When we define a basis in the tangent bundle, its dual in the cotangent bundle is also automatically defined. This dual basis is called −basisname and has the same cnumbers as the original. The notation for covariant bc−indices is

`{-a, -basisname}` : Basis index (`BIndex`), `a` is an abstract index.

`{i, -basisname}` : Component index (`CIndex`), `i` is a valid coordinate number.

Notice how a covariant b−index is −1 * (corresponding contravariant index), but this is not true for c−indices. This apparent inconsistency is a conscious choice: it allows us to use zero and negative cnumbers (the latter being common in treatments of angular momentum, tensor harmonics, etc.)

NOTE: In versions 0.3 and 0.4 we used b−indices like {a, polar}, {−a, polar} and c−indices like {1, polar} and {−1, polar}. This notational change is *not* backwards compatible. We apologize to those users of `xCoba`` who now need to change lots of signs in previously developed notebooks, but we feel that the use of this new notation will pay off.

Several functions are provided to check the validity.

```
In[63]:= IndicesOfVBundle[TangentM3]

Out[63]= {{a, b, c, d, e, f}, {f1, f2, f3}}

In[64]:= indexlist = { {a, polar}, {-a, -cartesian}, {w, polar}, {A, polar} ,
           {1, polar}, a , {1, -cartesian}, {a, spherical}, {5, polar}, {-1, -polar}};

In[65]:= $Bases

Out[65]= {polar, cartesian}

In[66]:= BIndexQ /@ indexlist

Out[66]= {True, True, False, False, False, False, False, False, False, False}

In[67]:= CIndexQ /@ indexlist

Out[67]= {False, False, False, False, True, False, True, False, False, False}
```

An extra function selects both basis and component indices,

```
In[68]:= BCIndexQ /@ indexlist

Out[68]= {True, True, False, False, True, False, True, False, False, False}
```

©2003−2004 José M. Martín−García

On ocassion, it is interesting to know whether an index is either an AIndex or a BIndex. Those are the only types that may be contracted, which makes them important for functions such as ContractBasis. We will call them 'contractible indices'.

```
In[69]:= ABIndexQ /@ indexlist

Out[69]= {True, True, False, False, False, True, False, False, False, False}
```

Notice that we can get a False output for several reasons: The basis, abstract index or cnumber might not exist or they might belong to different vbundles.

If we use these indices inside of tensors, *Mathematica* will display them according to BasisColor,

```
In[70]:= T[{1, -cartesian}, {2, -cartesian}] v[{a, polar}]

Out[70]= T_{1 2} v^a
```

Objects like those in the previous output can be seen as contractions of the tensor with some kind of basis object. xCoba ` provides Basis, a 1–covariant, 1–contravariant symmetric tensor. Its meaning depends on the type of indices used:

- Basis[AIndex, BCIndex] : Basis vector (or dual basis covector).
- Basis[AIndex, AIndex]: Identity tensor in the corresponding vbundle (turns into δ).
- Basis[BCIndex, BCIndex]:If the indices belong to different bases, it represents a basis change. If they belong to the same basis, we have a Kronecker delta (products of the elements of dual bases).

Basis is output as an e:

```
In[71]:= {Basis[a, {1, -polar}], Basis[a, -b],
          Basis[{a, cartesian}, {-b, -polar}], Basis[{a, cartesian}, {-b, -cartesian}],
          Basis[{1, polar}, {1, -polar}], Basis[{1, polar}, {2, -polar}]}

Out[71]= {e^a_1, δ^a_b, e^a_b, e^a_b, 1, 0}
```

If both indices are abstract, a different symbol (δ) is used. The reason for this is twofold: mathematical and computational. Basis has a lot of associated rules and definitions; if we added to them those of δ, we would have to perform many checks each time one of these objects appeared, thus slowing down the operation. Notice how Basis is Orderless. Indeed, some authors even write its indices stacked rather than staggered.

```
In[72]:= {Basis[a, {1, -polar}], Basis[{1, -polar}, a], Basis[-a, b], Basis[b, -a]}

Out[72]= {e^a_1, e^a_1, δ^b_a, δ^b_a}
```

A Basis object with two indices of the same character is converted into a metric tensor. If we have defined more than one metric, the first one is used (see xTensorDoc.nb for details)

```
In[73]:= Catch@Basis[-a, -b]

        MetricsOfVBundle::missing : There is no metric in TangentM3.
```

©2003–2004 José M. Martín–García

```
In[74]:= DefMetric[-1, metricg[-a, -b], CD, {";", "∇"}, PrintAs -> "g"]

          ** DefTensor: Defining symmetric metric tensor metricg[-a, -b].

          ** DefTensor: Defining antisymmetric tensor epsilonmetricg[a, b, c].

          ** DefCovD: Defining covariant derivative CD[-a].

          ** DefTensor: Defining vanishing torsion tensor TorsionCD[a, -b, -c].

          ** DefTensor: Defining symmetric Christoffel tensor ChristoffelCD[a, -b, -c].

          ** DefTensor: Defining Riemann tensor RiemannCD[-a, -b, -c, -d].

          ** DefTensor: Defining symmetric Ricci tensor RicciCD[-a, -b].

          ** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

          ** DefTensor: Defining Ricci scalar RicciScalarCD[].

          ** DefCovD:  Contractions of Ricci automatically replaced by RicciScalar.

          ** DefTensor: Defining symmetric Einstein tensor EinsteinCD[-a, -b].

          ** DefTensor: Defining vanishing Weyl tensor WeylCD[-a, -b, -c, -d].

          ** DefTensor: Defining symmetric TFRicci tensor TFRicciCD[-a, -b].

             Rules {1, 2} have been declared as DownValues for TFRicciCD.

          ** DefCovD:  Computing RiemannToWeylRules for dim 3

          ** DefCovD:  Computing RicciToTFRicci for dim 3

          ** DefCovD:  Computing RicciToEinsteinRules for dim 3

In[75]:= {Basis[-a, -b], Basis[{a, cartesian}, {b, cartesian}],
          Basis[{a, cartesian}, {b, polar}]}

Out[75]= {g_ab, g^ab, g^ab}

In[76]:= UndefMetric[metricg]

          ** UndefTensor: Undefined antisymmetric tensor epsilonmetricg

          ** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCD

          ** UndefTensor: Undefined symmetric Einstein tensor EinsteinCD

          ** UndefTensor: Undefined symmetric Ricci tensor RicciCD

          ** UndefTensor: Undefined Ricci scalar RicciScalarCD

          ** UndefTensor: Undefined Riemann tensor RiemannCD

          ** UndefTensor: Undefined symmetric TFRicci tensor TFRicciCD

          ** UndefTensor: Undefined vanishing torsion tensor TorsionCD

          ** UndefTensor: Undefined vanishing Weyl tensor WeylCD

          ** UndefCovD: Undefined covariant derivative CD

          ** UndefTensor: Undefined symmetric metric tensor metricg
```

Basis is considered to be a tensor whose master is Symbol, so it cannot be undefined:

©2003–2004 José M. Martín–García

```
In[77]:= xTensorQ[Basis]

Out[77]= True

In[78]:= MasterOf[Basis]

Out[78]= Symbol
```

The bracket of two basis vector fields is automatically converted into a torsion tensor (structure coefficients). This tensor is associated with the PD of the basis

```
In[79]:= Torsion[cartesian]

Out[79]= TorsionPDcartesian

In[80]:= Bracket[a][Basis[s, {1, -polar}], Basis[s, {2, -polar}]]

Out[80]= -T^a_{1 2}
```

If and only if the basis is coordinated, its torsion tensor will be `Zero`.

`Basis` and `Dir` act as inverses of each other

```
In[81]:= Dir[Basis[{a, polar}, -c]]

Out[81]= {a, polar}

In[82]:= Basis[{1, polar}, Dir[v[s]]]

Out[82]= v^1
```

An important issue is when to contract Basis objects with other tensors and with one another:

```
In[83]:= ContractBasis /@ {Basis[a, {-b, -polar}] Basis[-a, {c, cartesian}],
           Basis[a, {1, -polar}] Basis[-a, {1, polar}],
           Basis[a, {-b, -polar}] Basis[-c, {b, polar}]}

Out[83]= {e_b^c, 1, δ^a_c}
```

It would be desirable to automatize this behaviour sometimes. We can do it with the command `AutomaticBasisContraction-Start[]`:

`AutomaticBasisContractionStart`	Start automatic contraction of Basis objects
`AutomaticBasisContractionStop`	Stop automatic contraction of Basis objects

Contraction of Basis objects

```
In[84]:= AutomaticBasisContractionStart[]

In[85]:= {Basis[a, {-b, -polar}] Basis[-a, {c, cartesian}],
          Basis[a, {1, -polar}] Basis[-a, {1, polar}],
          Basis[a, {-b, -polar}] Basis[-c, {b, polar}]}

Out[85]= {e_b^c, 1, δ^a_c}
```

©2003–2004 José M. Martín–García

```
In[86]:= AutomaticBasisContractionStop[]
```

When the contraction is between a Basis object and a different tensor, it is only performed if the former is a formal Kronecker delta or an identity tensor,

```
In[87]:= {Basis[a, -b] T[-a, -c],
          Basis[{a, polar}, {-b, -polar}] v[{b, polar}],
          Basis[{a, polar}, -b] v[b],
          Basis[{a, cartesian}, {-b, -polar}] v[{b, polar}]}

Out[87]= {T_bc, v^a, e_b^a v^b, e^a_b v^b}
```

To force the contraction in the last two cases we need a new function (next section). Notice how the third contraction is equivalent to expressing a tensor in its components and the fourth represents a basis change.

We can define a special name for the elements of one particular basis.

```
In[88]:= FormatBasis[{1, polar}, "n"]

In[89]:= {Basis[-a, {1, polar}], Basis[-a, {2, polar}],
          Basis[{1, -cartesian}, {1, polar}], Basis[{2, polar}, {1, -cartesian}]}

Out[89]= {n_a, e_a^2, n_1, e_1^2}

In[90]:= FormatBasis[{2, polar}, "l"]

In[91]:= {Basis[-a, {1, polar}], Basis[-a, {2, polar}],
          Basis[{1, -cartesian}, {1, polar}], Basis[{2, polar}, {1, -cartesian}]}

Out[91]= {n_a, l_a, n_1, l_1}
```

We remove the special names:

```
In[92]:= FormatBasis[{1, polar}]
         FormatBasis[{2, polar}]
```

1.3. Bases on inner bundles and complex bases

Starting with version 0.9, `xTensor`` is capable of dealing with complex tensors and vector bundles. The function `DefBasis` has been extended accordingly and it now allows the user to define complex bases, through the option `Dagger`, taking the values `Real` or `Complex`. A basis on a real vbundle is real by default:

```
In[94]:= Dagger[polar]

Out[94]= polar
```

But let us see what happens when we define one on a complex bundle,

```
In[95]:= $Bases

Out[95]= {polar, cartesian}
```

©2003–2004 José M. Martín–García

```
In[96]:= DefBasis[comp, InnerC, {-1, +1}, Dagger → Complex, BasisColor → RGBColor[0, 0, 1]]

          ** DefCovD: Defining parallel derivative PDcomp[-a].

          ** DefTensor: Defining torsion tensor TorsionPDcomp[a, -b, -c].

          ** DefTensor: Defining
           non-symmetric Christoffel tensor ChristoffelPDcomp[a, -b, -c].

          ** DefTensor: Defining vanishing Riemann tensor RiemannPDcomp[-a, -b, -c, d].

          ** DefTensor: Defining vanishing Ricci tensor RicciPDcomp[-a, -b].

          ** DefTensor: Defining
           nonsymmetric AChristoffel tensor  AChristoffelPDcomp[A, -b, -C].

          ** DefTensor: Defining nonsymmetric AChristoffel tensor
           AChristoffelPDcomp†[A†, -b, -C†].

          ** DefTensor: Defining vanishing FRiemann tensor FRiemannPDcomp[-a, -b, -C, D].

          ** DefTensor: Defining
           vanishing FRiemann tensor FRiemannPDcomp†[-a, -b, -C†, D†].

          ** DefTensor: Defining antisymmetric +1 density etaUpcomp[A, B].

          ** DefTensor: Defining antisymmetric +1 density etaUpcomp†[A†, B†].

          ** DefTensor: Defining antisymmetric -1 density etaDowncomp[-A, -B].

          ** DefTensor: Defining antisymmetric -1 density etaDowncomp†[-A†, -B†].

In[97]:= $Bases

Out[97]= {polar, cartesian, comp, comp†}
```

A conjugate basis has also appeared on the conjugate manifold:

```
In[98]:= Dagger /@ {comp, comp†}

Out[98]= {comp†, comp}

In[99]:= VBundleOfBasis /@ {comp, comp†}

Out[99]= {InnerC, InnerC†}
```

Notice how negative cnumbers do not denote covariant indices:

```
In[100]:=
      {Basis[{-1, comp}, -a], Basis[{-1, comp†}, -a]}

Out[100]=
      {e_a^-1, e_a^-1}
```

The basis objects of both bases look the same, because

```
In[101]:=
      Dagger[Basis]

Out[101]=
      Basis
```

©2003–2004 José M. Martín–García

```
In[102]:=
Dagger[Basis[{-1, comp}, -a]]

Out[102]=
e_a^-1

In[103]:=
InputForm[%]

Out[103]//InputForm=
Basis[-a, {-1, comp†}]
```

and their cnumbers are also defined to be the same. This can be changed, with the new function `DaggerBCIndex`. Remember how in `xTensor`` the conjugates of indices were given by

```
In[104]:=
DaggerIndex /@ {a, b, A, B}

Out[104]=
{a, b, A†, B†}
```

But we cannot change the definition associated to `DaggerIndex[{1, complex}]`, because `complex` is too deep to be given an upvalue. Let us see how we can use `DaggerCIndex` to solve this with an example. We want to complexify a real basis:

```
In[105]:=
DefBasis[test, TangentM3, {1, 2, 3}]

** DefCovD: Defining parallel derivative PDtest[-a].

** DefTensor: Defining torsion tensor TorsionPDtest[a, -b, -c].

** DefTensor: Defining
 non-symmetric Christoffel tensor ChristoffelPDtest[a, -b, -c].

** DefTensor: Defining vanishing Riemann tensor RiemannPDtest[-a, -b, -c, d].

** DefTensor: Defining vanishing Ricci tensor RicciPDtest[-a, -b].

** DefTensor: Defining antisymmetric +1 density etaUptest[a, b, c].

** DefTensor: Defining antisymmetric -1 density etaDowntest[-a, -b, -c].

In[106]:=
?DaggerCIndex

DaggerCIndex[basis, cindex] returns the conjugated index to the
  cindex (assumed to belong to basis), in general a Dir expression.

In[107]:=
test /: DaggerCIndex[test, {1, test}] = {2, test};
test /: DaggerCIndex[test, {2, test}] = {1, test};
test /: DaggerCIndex[test, {1, -test}] = {2, -test};
test /: DaggerCIndex[test, {2, -test}] = {1, -test};
```

And now

```
In[111]:=
Dagger /@ {Basis[{1, test}, -a], Basis[{2, -test}, a]}

Out[111]=
{e_a^2, e^a_2}
```

©2003–2004 José M. Martín–García

If we define a complex tensor on M3 we can see that this works

```
In[112]:=
DefTensor[J[a], M3, Dagger → Complex]

    ** DefTensor: Defining tensor J[a].

    ** DefTensor: Defining tensor J†[a].

In[113]:=
Dagger[J[{2, test}]]

Out[113]=
J†[1]
```

We can even establish more complicated relations, if we remember that Basis and Dir act as inverses of each other

```
In[114]:=
test /: DaggerCIndex[test, {1, test}] = Dir[I Basis[{2, test}, -a]];
test /: DaggerCIndex[test, {2, test}] = Dir[I Basis[{1, test}, -a]];

In[116]:=
Dagger[J[{2, test}]]

Out[116]=
i J†[1]

In[117]:=
UndefBasis[test];

    ** UndefTensor: Undefined non-symmetric Christoffel tensor ChristoffelPDtest

    ** UndefTensor: Undefined vanishing Ricci tensor RicciPDtest

    ** UndefTensor: Undefined vanishing Riemann tensor RiemannPDtest

    ** UndefTensor: Undefined torsion tensor TorsionPDtest

    ** UndefCovD: Undefined parallel derivative PDtest

    ** UndefTensor: Undefined antisymmetric +1 density etaUptest

    ** UndefTensor: Undefined antisymmetric -1 density etaDowntest

    ** Undefined basis test
```

1.4 BasisArray

`BasisArray`	List of basis vectors

Listing basis objects

The function `BasisArray` returns a list of all basis vectors of a given basis

©2003–2004 José M. Martín–García

In[118]:=
```
BasisArray[cartesian][a]
```

Out[118]=
$$\{e^a{}_0,\ e^a{}_1,\ e^a{}_2\}$$

In[119]:=
```
BasisArray[comp][A]
```

Out[119]=
$$\{e^A{}_{-1},\ e^A{}_1\}$$

If we give several indices, the product basis is displayed

In[120]:=
```
BasisArray[polar, polar][a, b]
```

Out[120]=
$$\{\{e^a{}_0\ e^b{}_0,\ e^a{}_0\ e^b{}_1,\ e^a{}_0\ e^b{}_2\},\ \{e^a{}_1\ e^b{}_0,\ e^a{}_1\ e^b{}_1,\ e^a{}_1\ e^b{}_2\},\ \{e^a{}_2\ e^b{}_0,\ e^a{}_2\ e^b{}_1,\ e^a{}_2\ e^b{}_2\}\}$$

We can mix bases

In[121]:=
```
BasisArray[polar, cartesian, comp][a, c, -B] // MatrixForm
```

Out[121]//MatrixForm=
$$\begin{pmatrix} \begin{pmatrix} e^a{}_0\ e_B{}^{-1}\ e^c{}_0 \\ e^a{}_0\ e_B{}^1\ e^c{}_0 \end{pmatrix} & \begin{pmatrix} e^a{}_0\ e_B{}^{-1}\ e^c{}_1 \\ e^a{}_0\ e_B{}^1\ e^c{}_1 \end{pmatrix} & \begin{pmatrix} e^a{}_0\ e_B{}^{-1}\ e^c{}_2 \\ e^a{}_0\ e_B{}^1\ e^c{}_2 \end{pmatrix} \\ \begin{pmatrix} e^a{}_1\ e_B{}^{-1}\ e^c{}_0 \\ e^a{}_1\ e_B{}^1\ e^c{}_0 \end{pmatrix} & \begin{pmatrix} e^a{}_1\ e_B{}^{-1}\ e^c{}_1 \\ e^a{}_1\ e_B{}^1\ e^c{}_1 \end{pmatrix} & \begin{pmatrix} e^a{}_1\ e_B{}^{-1}\ e^c{}_2 \\ e^a{}_1\ e_B{}^1\ e^c{}_2 \end{pmatrix} \\ \begin{pmatrix} e^a{}_2\ e_B{}^{-1}\ e^c{}_0 \\ e^a{}_2\ e_B{}^1\ e^c{}_0 \end{pmatrix} & \begin{pmatrix} e^a{}_2\ e_B{}^{-1}\ e^c{}_1 \\ e^a{}_2\ e_B{}^1\ e^c{}_1 \end{pmatrix} & \begin{pmatrix} e^a{}_2\ e_B{}^{-1}\ e^c{}_2 \\ e^a{}_2\ e_B{}^1\ e^c{}_2 \end{pmatrix} \end{pmatrix}$$

■ 2. Manipulating basis indices: ContractBasis, SeparateBasis, ToBasis

We ended the previous section with an open problem: what to do with expressions such as

In[122]:=
```
{Basis[{a, polar}, -b] v[b], Basis[{a, cartesian}, {-b, -polar}] v[{b, polar}],
 Basis[a, {-b, -polar}] v[{b, polar}]}
```

Out[122]=
$$\{e_b{}^a\ v^b,\ e^a{}_b\ v^b,\ e^a{}_b\ v^b\}$$

The first one represents the expansion of a tensor into its components and the second one a basis change. These operations are seemingly different: we are alternatively contracting basis indices and abstract indices. They are all, however, encompassed by one single function in `xCoba`: `ContractBasis`.

Conversely, we may want to go from contracted expressions to ones with explicit `Basis` objects. And we may also need to introduce additional `Basis` objects in order to take care of a basis change:

©2003–2004 José M. Martín–García

```
In[123]:=
{v[{a, polar}] == v[b] Basis[-b, {a, polar}],
 v[a] == Basis[a, {-b, -polar}] v[{b, polar}],
 v[{a, polar}] == v[{b, cartesian}] Basis[{a, polar}, {-b, -cartesian}]}
```

```
Out[123]=
{v^a == e_b^a v^b, v^a == e^a_b v^b, v^a == e^a_b v^b}
```

In the first example, we are simply extracting the Basis object from the tensor, so to speak; but in the second and third cases we are introducing new `Basis` objects, either to express a tensor in terms of its components or to change their basis. Again, all these operations are performed by just one function: `SeparateBasis`.

2.1. IndicesOf

In `xTensor`, the main user level function to find indices in an expression was `IndicesOf[selectors][expression]`,

```
In[124]:=
DefTensor[TT[-a, -b, c, d, -e], M3]
DefTensor[U[-a, -b, c, d], M3]
DefTensor[S[-A, B], M3, Dagger → Complex]
```

```
** DefTensor: Defining tensor TT[-a, -b, c, d, -e].

** DefTensor: Defining tensor U[-a, -b, c, d].

** DefTensor: Defining tensor S[-A, B].

** DefTensor: Defining tensor S†[-A†, B†].
```

```
In[127]:=
expr = TT[{1, -polar}, -b, {2, cartesian}, c, -a] Basis[{1, -polar}, a] +
  Basis[a, {1, -polar}] Basis[-d, {2, cartesian}] U[-a, -b, d, {e, polar}]
   Basis[{-e, -polar}, c] S[{-A, -comp}, B] S[-B, {A, comp}]
```

```
Out[127]=
e^a_1 TT_1b^2c_a + e^a_1 e^c_e e_d^2 S_B^A S_A^B U_ab^de
```

```
In[128]:=
IndicesOf[Free][expr]
```

```
Out[128]=
{-b, c}
```

```
In[129]:=
IndicesOf[TangentM3][expr]
```

```
Out[129]=
{a, {1, -polar}, -b, {2, cartesian}, c, -a, {-e, -polar}, -d, d, {e, polar}}
```

```
In[130]:=
IndicesOf[InnerC][expr]
```

```
Out[130]=
{-B, {A, comp}, {-A, -comp}, B}
```

```
In[131]:=
IndicesOf[U, Free][expr]
```

```
Out[131]=
{-a, -b, d, {e, polar}}
```

©2003–2004 José M. Martín–García

```
In[132]:=
IndicesOf[Free, U][expr]

Out[132]=
{-b}

In[133]:=
IndicesOf[{U, Free}][expr]

Out[133]=
{-a, -b, d, {e, polar}, -b, c}
```

(See xTensorDoc.nb, section 9.1). `xCoba`` adds more possibilities. We can now use the types `BIndex` and `CIndex` as selectors, or search for a particular basis

```
In[134]:=
IndicesOf[{BIndex, CIndex}][expr]

Out[134]=
{{-e, -polar}, {A, comp}, {-A, -comp}, {e, polar}, {1, -polar}, {2, cartesian}}

In[135]:=
IndicesOf[polar][expr]

Out[135]=
{{1, -polar}, {-e, -polar}, {e, polar}}
```

Consider now the following example

```
In[136]:=
IndicesOf[Basis, polar][expr]

Out[136]=
{{1, -polar}, {-e, -polar}}

In[137]:=
IndicesOf[Basis[polar]][expr]

Out[137]=
{a, c, {1, -polar}, {-e, -polar}}
```

The first selects all the indices appearing in Basis objects and then selects from those only the ones that belong to polar. The second selects only those indices that belong to a Basis object of the polar basis, but returns the pair, not only the `BCIndex`.

```
In[138]:=
IndicesOf[Basis[polar]][Basis[{a, polar}, {-b, -cartesian}]]

Out[138]=
{{a, polar}, {-b, -cartesian}}
```

This function will be used heavily to control the behaviour of ContractBasis and SeparateBasis.

2.2. ContractBasis

`ContractBasis` is responsible for all contractions of `Basis` with any other object.

©2003–2004 José M. Martín–García

`ContractBasis[expr, indices]` dummy indices	Perform the contractions of `Basis` objects involving the specified

Contraction of Basis objects with tensors.

We have complete control over which dummy indices should be eliminated. We can select them in several ways, all of which are eventually converted into a list of indices with head `IndexList`:

- `IndexList[...]`,

be careful not to use the normal *Mathematica* `List`, which could be mistaken for a `BCIndex`. `IndicesOf[selectors]` as a second argument is treated as `IndicesOf[selectors][expr]` and gives great flexibility, but it is sometimes too long for simple cases. To handle them, several shorthands are defined:

- `ContractBasis[expr,basis]` = `ContractBasis[expr,IndicesOf[basis]]`,
- `ContractBasis[expr,vbundle]` = `ContractBasis[expr,IndicesOf[vbundle]]`,
- `ContractBasis[expr,tensor]` = `ContractBasis[expr,IndicesOf[tensor]]`,
- `ContractBasis[expr,index]` = `ContractBasis[expr,IndexList[index]]`.

It is important to keep in mind that the second argument gives the list of indices to be contracted, not their pairs in the Basis object.

All of this is best illustrated through some examples:

```
In[139]:=
expr
```

```
Out[139]=
e^a_1 TT_1b^2c_a + e^a_1 e^c_e e_d^2 S_B^A S_A^B U_ab^de
```

```
In[140]:=
expr2 = Basis[{-e, -cartesian}, b] Basis[-A, {1, comp}] S[-B, A] v[{e, cartesian}]
```

```
Out[140]=
e_A^1 e^b_e S_B^A v^e
```

The function called with just one argument performs all the possible contractions.

```
In[141]:=
ContractBasis[expr]
```

```
Out[141]=
TT_1b^2c_1 + S_B^A S_A^B U_1b^2c
```

If we choose to give the indices directly, we can specifiy either element of the pair

```
In[142]:=
{{ContractBasis[expr2, A], ContractBasis[expr2, -A]},
  {ContractBasis[expr2, {-e, -cartesian}], ContractBasis[expr2, {e, cartesian}]}} //
 TableForm
```

```
Out[142]//TableForm=
e^b_e S_B^1 v^e        e^b_e S_B^1 v^e
e_A^1 S_B^A v^b        e_A^1 S_B^A v^b
```

©2003–2004 José M. Martín–García

Nothing happens if we give an invalid index

```
In[143]:=
ContractBasis[expr2, C]
```

```
Out[143]=
e_A^1 e^b_e S_B^A v^e
```

More simple examples

```
In[144]:=
ContractBasis[expr, IndicesOf[AIndex]]
```

```
Out[144]=
TT_1b^2c_1 + e^c_e S_B^A S_A^B U_1b^2e
```

```
In[145]:=
ContractBasis[expr, IndicesOf[TT, U]]
```

```
Out[145]=
TT_1b^2c_1 + e^c_e e_d^2 S_B^A S_A^B U_1b^de
```

```
In[146]:=
ContractBasis[expr, IndicesOf[{TT, U}]]
```

```
Out[146]=
TT_1b^2c_1 + S_B^A S_A^B U_1b^2c
```

```
In[147]:=
ContractBasis[expr2, InnerC]
```

```
Out[147]=
e^b_e S_B^1 v^e
```

```
In[148]:=
ContractBasis[expr, IndexList[a, -d]]
```

```
Out[148]=
TT_1b^2c_1 + e^c_e S_B^A S_A^B U_1b^2e
```

```
In[149]:=
ContractBasis[expr, polar]
```

```
Out[149]=
e^a_1 TT_1b^2c_a + e^a_1 e_d^2 S_B^A S_A^B U_ab^dc
```

Notice how only the index `{-e, -polar}` has been supressed. Nothing has been done to `a`, even though its pair in the Basis object belongs to `polar`. If we want to act on that index, we can use

```
In[150]:=
ContractBasis[expr, IndicesOf[Basis[polar]]]
```

```
Out[150]=
TT_1b^2c_1 + e_d^2 S_B^A S_A^B U_1b^dc
```

©2003–2004 José M. Martín–García

to contract everything that appears in a Basis object that has at least one polar index. Or

```
In[151]:=
ContractBasis[expr, IndicesOf[Basis[polar], AIndex]]

Out[151]=
TT1 b2 c1 + ec e ed2 SBA SAB U1 bd e
```

to contract only abstract indices whose pair in `Basis` belongs to `polar`. Notice that some specifications can be redundant; `IndicesOf[Basis, x]` is equivalent to `IndicesOf[x]`, because all the indices that can be contracted by this function already have to appear in `Basis`.

Contraction of `Basis` objects can be performed with `ContractBasis`, or automatized:

```
In[152]:=
Basis[a, {-b, -cartesian}] Basis[-a, {c, polar}]

Out[152]=
ea c ea b

In[153]:=
ContractBasis[%]

Out[153]=
eb c

In[154]:=
AutomaticBasisContractionStart[]

In[155]:=
Basis[a, {-b, -cartesian}] Basis[-a, {c, polar}]

Out[155]=
eb c

In[156]:=
AutomaticBasisContractionStop[]
```

The default behaviour of `ContractBasis` disregards contractions involving derivatives. We must use the option `OverDerivatives`:

```
In[157]:=
DefCovD[CD[-a], {"#", "D"}]

** DefCovD: Defining covariant derivative CD[-a].

** DefTensor: Defining vanishing torsion tensor TorsionCD[a, -b, -c].

** DefTensor: Defining symmetric Christoffel tensor ChristoffelCD[a, -b, -c].

** DefTensor: Defining Riemann tensor
 RiemannCD[-a, -b, -c, d]. Antisymmetric only in the first pair.

** DefTensor: Defining non-symmetric Ricci tensor RicciCD[-a, -b].

** DefCovD:  Contractions of Riemann automatically replaced by Ricci.
```

©2003–2004 José M. Martín–García

In[158]:=
```
Basis[{1, polar}, -b] CD[-a][S[c, b]]
```

Out[158]=
$$e_b^{\ 1}\ (D_a\ S^{cb})$$

In[159]:=
```
ContractBasis[%]
```

Out[159]=
$$e_b^{\ 1}\ (D_a\ S^{cb})$$

In[160]:=
```
ContractBasis[%, OverDerivatives → True]
```

```
    ** DefTensor: Defining tensor ChristoffelCDPDpolar[a, -b, -c].
```

Out[160]=
$$\Gamma[D,\mathcal{D}]^{1}{}_{ab}\ S^{cb} + D_a\ S^{c\,1}$$

A new Christoffel tensor is automatically defined. We shall see more about the interplay between bases and derivatives in Section 5. Any type of derivate exhibits the same behaviour,

In[161]:=
```
lieexpr = LieD[v[a]][T[-a, -b]] Basis[{1, -polar}, a] Basis[{3, -polar}, b]
```

Out[161]=
$$e^a_{\ 1}\ e^b_{\ 3}\ (\mathcal{L}_v\ T_{ab})$$

In[162]:=
```
{ContractBasis[lieexpr], ContractBasis[lieexpr, OverDerivatives → True]} // TableForm
```

Out[162]//TableForm=
$$e^a_{\ 1}\ e^b_{\ 3}\ (\mathcal{L}_v\ T_{ab})$$
$$\mathcal{L}_v\ T_{1\,3} + T_{a\,3}\ (\mathcal{D}_1\ v^a) + T_{1\,b}\ (\mathcal{D}_3\ v^b)$$

In[163]:=
```
paramexpr = OverDot[T[-a, -b]] Basis[{1, -polar}, a] Basis[{3, -polar}, b]
```

Out[163]=
$$e^a_{\ 1}\ e^b_{\ 3}\ \dot{T}_{ab}$$

In[164]:=
```
{ContractBasis[paramexpr],
  ContractBasis[paramexpr, OverDerivatives → True]} // TableForm
```

Out[164]//TableForm=
$$e^a_{\ 1}\ e^b_{\ 3}\ \dot{T}_{ab}$$
$$\dot{T}_{1\,3} - e^{\dot{a}}_{\ 1}\ T_{a\,3} - e^{\dot{b}}_{\ 3}\ T_{1\,b}$$

In[165]:=
```
expr2 =.
lieexpr =.
paramexpr =.
```

©2003–2004 José M. Martín–García

In[168]:=
```
UndefCovD[CD]
```

```
** UndefTensor: Undefined symmetric Christoffel tensor ChristoffelCD

** UndefTensor: Undefined tensor ChristoffelCDPDpolar

** UndefTensor: Undefined non-symmetric Ricci tensor RicciCD

** UndefTensor: Undefined Riemann tensor RiemannCD

** UndefTensor: Undefined vanishing torsion tensor TorsionCD

** UndefCovD: Undefined covariant derivative CD
```

The input expression is automatically expanded (with `Expand`) before applying `ContractBasis`. It would be better, in principle, to contract the bases only locally, but this is not yet implemented.

At the moment, the user cannot specify in which order the indices should be contracted. An option `ContractFirst` is planned but not yet coded.

2.3. SeparateBasis

`SeparateBasis[basis][expr, indices]`	Expand the selected indices of a given expression in the specified basis

Expansion of an expression in a given basis.

`SeparateBasis` works in a similar way to `ContractBasis`, but now we have two brackets. In the second one we can specify which indices should be 'separated' from their original tensor. In the first one we say what should remain in their place (either `AIndex` or a basis).

In[169]:=
```
expr = TT[{1, -polar}, -b, {2, cartesian}, c, -a] Basis[{1, -polar}, a] +
  Basis[a, {1, -polar}] Basis[-d, {2, cartesian}]
   U[-a, -b, d, {e, polar}] Basis[{-e, -polar}, c]
```

Out[169]=
$$e^{a}_{1}\ TT_{1\,b}{}^{2\,c}{}_{a} + e^{a}_{1}\ e^{c}_{e}\ e_{d}{}^{2}\ U_{ab}{}^{d\,e}$$

In[170]:=
```
cexpr = ContractBasis[expr]
```

Out[170]=
$$TT_{1\,b}{}^{2\,c}{}_{1} + U_{1\,b}{}^{2\,c}$$

In[171]:=
```
SeparateBasis[AIndex][cexpr] // ScreenDollarIndices
```

Out[171]=
$$e^{a}_{1}\ e_{d}{}^{2}\ e^{e}_{1}\ TT_{ab}{}^{dc}{}_{e} + e^{a}_{1}\ e_{d}{}^{2}\ U_{ab}{}^{dc}$$

In[172]:=
```
Simplify[%]
```

Out[172]=
$$e^{a}_{1}\ e_{d}{}^{2}\ \left(e^{e}_{1}\ TT_{ab}{}^{dc}{}_{e} + U_{ab}{}^{dc}\right)$$

©2003–2004 José M. Martín–García

But we may also want to perform a change of basis. This is done as follows

```
In[173]:=
SeparateBasis[cartesian][cexpr] // ScreenDollarIndices
```

```
Out[173]=
e_b^d e^c_f e_1^a e_1^f1 e^2_e TT_ad^ef_f1 + e_b^d e^c_f e_1^a e^2_e U_ad^ef
```

```
In[174]:=
Simplify[%]
```

```
Out[174]=
e_b^d e^c_f e_1^a e^2_e (e_1^f1 TT_ad^ef_f1 + U_ad^ef)
```

Now all tensor indices belong to the specified basis.

We have the same syntax as in `ContractBasis` for the second argument

```
In[175]:=
$PrePrint = ScreenDollarIndices;
```

```
In[176]:=
SeparateBasis[AIndex][cexpr, {4, cartesian}]
```

```
Out[176]=
TT_1b^2c_1 + U_1b^2c
```

```
In[177]:=
SeparateBasis[AIndex][cexpr, IndicesOf[polar]]
```

```
Out[177]=
e^a_1 e^d_1 TT_ab^2c_d + e^a_1 U_ab^2c
```

```
In[178]:=
SeparateBasis[AIndex][cexpr, IndicesOf[CIndex]]
```

```
Out[178]=
e^a_1 e_d^2 e^e_1 TT_ab^dc_e + e^a_1 e_d^2 U_ab^dc
```

When separating component indices, the function selects only those with valid coordinate numbers,

```
In[179]:=
v[{1, polar}] v[{2, polar}] v[{7, polar}] // SeparateBasis[AIndex]
```

```
Out[179]=
e_a^1 e_b^2 v^a v^b v^7
```

We must be careful when working with scalar functions of scalars, lest we get meaningless outputs, such as powers of tensor products :

```
In[180]:=
v[{1, polar}] // SeparateBasis[AIndex]
```

```
Out[180]=
e_a^1 v^a
```

©2003–2004 José M. Martín–García

```
In[181]:=
v[{1, polar}] v[{1, polar}] // SeparateBasis[AIndex]

Out[181]=
v^1 2

In[182]:=
%% ^2

Out[182]=
e_f$1109^1 2 v^f$1109 2
```

The safe way to manipulate such expressions is via the `Scalar` head,

```
In[183]:=
Scalar[v[{1, polar}]] Scalar[v[{1, polar}]]

Out[183]=
Scalar[v^1]^2

In[184]:=
% // SeparateBasis[cartesian]

Out[184]=
Scalar[v^1]^2

In[185]:=
% /. Scalar[scalar_] :→ Scalar[SeparateBasis[cartesian][scalar]]

Out[185]=
Scalar[e^1_a v^a]^2
```

If a component index is repeated, we may need to apply `SeparateBasis` several times

```
In[186]:=
expr4 = T[{1, -polar}, {1, -polar}]

Out[186]=
T_1 1

In[187]:=
% // SeparateBasis[AIndex]

Out[187]=
e^a_1 e^b_1 T_ab
```

It works, but if we specify the index,

```
In[188]:=
SeparateBasis[AIndex][expr4, {1, -polar}]

Out[188]=
e^a_1 T_a 1
```

©2003–2004 José M. Martín–García

`{1, -polar}` appears twice in the same tensor, so we need to reapply `SeparateBasis` (or pass it twice to the function)

```
In[189]:=
SeparateBasis[AIndex][%, {1, -polar}]

Out[189]=
e^a_1 e^b_1 T_ab

In[190]:=
SeparateBasis[AIndex][expr4, IndexList[{1, -polar}, {1, -polar}]]

Out[190]=
e^a_1 e^b_1 T_ab
```

2.4. ToBasis

`FreeToBasis[basis][expr,indices]`	Convert free indices into basis indices
`FreeToBasis[basis][expr,indices]`	Change the basis of pairs of dummies
`ToBasis[basis][expr,indices]`	Replace the indices in an expression by indices in the given basis

Safe replacement of basis indices.

We may want to replace the indices in an expression by indices in a different basis. Sometimes this is easy enough

```
In[191]:=
SeparateBasis[polar][v[a] v[-a]]

Out[191]=
e_a^b e^a_c v_b v^c

In[192]:=
% // ContractBasis

Out[192]=
v_a v^a
```

And it may seem that the `ReplaceIndex` function could take care of the more difficult situations. But we must watch for pitfalls such as

```
In[193]:=
ReplaceIndex[v[a] PD[-b][v[-a]], {a → {a, polar}, -a → {-a, -polar}}]

Out[193]=
v^a ∂_b v_a
```

This is wrong, because

$$v^b \nabla_a v_b = (v^c e^b{}_c) \nabla_a (e_b{}^d v_d) \neq v^b \nabla_a v_b$$

We can arrive at the correct result via a roundabout way, with `Contract` and `SeparateBasis`

```
In[194]:=
SeparateBasis[polar][v[a] PD[-b][v[-a]]] // ScreenDollarIndices

Out[194]=
e^a_c v^c (e_b^e Γ[𝒟]^d_e a v_d + e_a^d e_b^f ∂_f v_d)
```

©2003–2004 José M. Martín–García

```
In[195]:=
ContractBasis[%]

Out[195]=
Γ[𝒟]^d_{b c} v^c v_d + v^d ∂_b v_d
```

The functions *ToBasis automate this process

```
In[196]:=
DummyToBasis[polar][v[a] PD[-b][v[-a]]] // Simplification

Out[196]=
Γ[𝒟]^a_{b c} v_a v^c + v^a ∂_b v_a
```

Notice how the derivative of the Basis object has been replaced by the appropriate Christoffel. Further examples:

```
In[197]:=
expr = S[-A, B] v[a] v[c] v[-c]

Out[197]=
S_A^B v^a v_c v^c

In[198]:=
FreeToBasis[cartesian][%]

Out[198]=
S_A^B v_c v^c v^a

In[199]:=
ToBasis[cartesian][%]

Out[199]=
S_A^B v^a v_b v^b

In[200]:=
FreeToBasis[comp][%]

Out[200]=
S_A^B v^a v_b v^b

In[201]:=
ToBasis[polar][%, {a, cartesian}]

Out[201]=
S_A^B v^a v_b v^b
```

Many times we have expressions such as

```
In[202]:=
T[{-a, -polar}, {-b, -polar}] v[{a, polar}] v[{b, polar}]

Out[202]=
T_{a b} v^a v^b
```

©2003–2004 José M. Martín–García

where we had defined T_{ab} to be antisymmetric. Therefore this is zero:

```
In[203]:=
Simplification[%]

Out[203]=
0
```

However, if we have

```
In[204]:=
v[-a] v[a] - v[{-a, -polar}] v[{a, polar}]

Out[204]=
v_a v^a - v_a v^a

In[205]:=
Simplification[%]

Out[205]=
v_a v^a - v_a v^a
```

we have to change the whole expression to a single basis first

```
In[206]:=
ToBasis[AIndex][%]

Out[206]=
0
```

■ 3. Parallel derivatives and Christoffel tensors

We begin by remembering that each basis has its own parallel derivative

```
In[207]:=
{PDpolar[-a][v[b]], PDcartesian[-a][T[-b, -c]]}

Out[207]=
{D_a v^b, D_a T_bc}
```

defined so that

```
In[208]:=
PDpolar[-a][Basis[{b, polar}, -c]]

Out[208]=
0
```

This derivative is always flat, and has a vanishing Riemann

```
In[209]:=
RiemannPDpolar[-a, -b, -c, -d]

Out[209]=
0
```

©2003–2004 José M. Martín–García

But it has torsion, unless the basis is coordinated,

```
In[210]:=
TorsionPDcartesian[a, -b, -c]

Out[210]=
T^a_bc

In[211]:=
Bracket[a][Basis[s, {1, -polar}], Basis[s, {3, -polar}]]

Out[211]=
-T^a_1 3
```

In `xTensor`` we can define the Christoffel tensor connecting two covariant derivatives (*cf.* Section 6 of xTensor–Doc.nb). In particular, this works with the PD of our bases

```
In[212]:=
Christoffel[PDpolar, PDcartesian][a, -b, -c]

    ** DefTensor: Defining tensor ChristoffelPDcartesianPDpolar[a, -b, -c].

Out[212]=
-Γ[𝒟, 𝒟]^a_bc
```

Notice how the Christoffel connecting any covariant derivative to PD is defined with the derivative, in our case by `DefBasis`.

```
In[213]:=
Christoffel[PDpolar, PD][a, -b, -c]

Out[213]=
Γ[𝒟]^a_bc
```

Any derivative of any `Basis` object can be translated into a component of Christoffel tensors relating that derivative to the PDs of the bases involved

```
In[214]:=
PD[-a][Basis[{1, polar}, -b]]

Out[214]=
Γ[𝒟]^1_ab

In[215]:=
PD[-a][Basis[{a, polar}, {-b, -cartesian}]]

Out[215]=
-Γ[𝒟, 𝒟]^a_a b

In[216]:=
PD[-a][Basis[{b, polar}, {-c, -polar}]]

Out[216]=
0
```

©2003–2004 José M. Martín–García

With another parallel derivative,

```
In[217]:=
PDpolar[-a][Basis[{1, -polar}, b]]

Out[217]=
0

In[218]:=
PDpolar[-a][Basis[{1, -cartesian}, b]]

Out[218]=
-Γ[𝒟,𝒟]^b_a 1
```

Complex bases have FRiemann and AChristoffel tensors, as explained in xTensorDoc, section 6.8.

```
In[219]:=
?AChristoffelPDcomp

    Global`AChristoffelPDcomp

    Dagger[AChristoffelPDcomp] ^= AChristoffelPDcomp†

    DependenciesOfTensor[AChristoffelPDcomp] ^= {M3}

    Info[AChristoffelPDcomp] ^= {nonsymmetric AChristoffel tensor , }

    MasterOf[AChristoffelPDcomp] ^= PDcomp

    PrintAs[AChristoffelPDcomp] ^= A[𝒟]

    ServantsOf[AChristoffelPDcomp] ^= {AChristoffelPDcomp†}

    SlotsOfTensor[AChristoffelPDcomp] ^= {InnerC, -TangentM3, -InnerC}

    SymmetryGroupOfTensor[AChristoffelPDcomp] ^= StrongGenSet[{}, GenSet[]]

    TensorID[AChristoffelPDcomp] ^= {AChristoffel, PDcomp, PD}

    xTensorQ[AChristoffelPDcomp] ^= True
```

The FRiemann is also zero

```
In[220]:=
FRiemannPDcomp[-A, -B, -C, -D]

Out[220]=
0
```

Derivatives of complex bases are also replaced by the corresponding Christoffels

```
In[221]:=
PD[-a][Basis[C, {-A, -comp}]]

Out[221]=
-A[𝒟]^C_a A
```

©2003–2004 José M. Martín–García

```
In[222]:=
PD[-a][Basis[C†, {-A†, -comp}]]

Out[222]=
-A[D] †^C†
       a A†

In[223]:=
PD[-a][Basis[b, {-c, -comp}]]

Out[223]=
-Γ[D]^b
     a c

In[224]:=
PDpolar[-b][Basis[C, {-A, -comp}]]

    ** DefTensor: Defining tensor AChristoffelPDcompPDpolar[A, -b, -C].

    ** DefTensor: Defining tensor AChristoffelPDcompPDpolar†[A†, -b, -C†].

Out[224]=
-A[D,D]^C
     b A
```

Both basis indices must belong to the same vbundle:

```
In[225]:=
Catch@PD[-a][Basis[B, {-c, -comp}]]

Christoffel::error :
 Indices #1 and #3 of (A)Christoffel must belong to the same vector bundle.
```

3.1. Ricci rotation coefficients

Ricci rotation coefficients are not supported, because they do not add any new information. `xCoba`' limits itself to converting `RicciRotation[covd,basis]` into `Christoffel[covd, PDofBasis[basis]]`.

```
In[226]:=
RicciRotation[PDpolar, cartesian][a, -b, -c]

Out[226]=
-Γ[D,D]^a
     bc

In[227]:=
RicciRotation[PDcartesian][a, -b, -c]

Out[227]=
Γ[D]^a
     bc
```

■ 4. Tensor densities; η and ϵ tensors and determinants

4.1. The η tensors and the Jacobian

`EtaUp`	Define a basis
`EtaDown`	VBundle on which a basis lives
`Jacobian`	Parallel derivative associated to the given basis
`WeightOf`	Weight of an expression as a linear combination of basis names

©2003–2004 José M. Martín–García

Definition of a basis.

The η tensors represent the totally antisymmetric products of all basis (co)vectors:

$$\tilde{\eta}^{a_1 \cdots a_n} = n!\, e^{[a_1}{}_1 \cdots e^{a_n]}{}_n, \qquad \underset{\sim}{\eta}_{a_1 \cdots a_n} = n!\, e_{[a_1}{}^1 \cdots e_{a_n]}{}^n$$

In `xCoba`, these objects are automatically defined for each basis.

```
In[228]:=
{etaUppolar[a, b, c], etaDownpolar[-a, -b, -c]}
```

```
Out[228]=
{η̃^abc, η_abc}
```

The basis is marked with an over or undertilde, whose significance we shall see below. Their components in the natural basis are very simple:

```
In[229]:=
{etaUppolar[{1, polar}, {2, polar}, {3, polar}],
 etaUppolar[{1, polar}, {3, polar}, {2, polar}],
 etaUppolar[{3, polar}, {1, polar}, {2, polar}],
 etaUppolar[{1, polar}, {1, polar}, {1, polar}]}
```

```
Out[229]=
{1, -1, 1, 0}
```

And they are related to the generalised Kronecker delta:

```
In[230]:=
etaUppolar[a, b, c] etaDownpolar[-d, -e, -f]
```

```
Out[230]=
δ^abc_def
```

The presence of a metric provides additional relations, as we shall see later. Each basis has a different pair of η tensors but they are all related through the Jacobians of the transformations. Because of this, η are generally considered not tensors, but tensor densities with weight = ± 1. In `xCoba` we consider them tensors, but dependent on a reference basis. If we change the reference basis (e.g., from etaUppolar to etaUpcartesian), we have to include Jacobians in the transformation. However, if we don't change the reference basis (e.g., from etaUppolar[{a,polar},{b, polar}, {c,polar}] to etaUppolar[{a,cartesian}, {b,cartesian}, {c, cartesian}]) they transform as tensors. This is analogous to our treatment of Christoffel tensors. Consider this example

```
In[231]:=
etaUppolar[a, b, c] etaDowncartesian[-d, -e, -f]
```

```
Out[231]=
η_def η̃^abc
```

This product is a density of weight +1 if we change the basis polar and of weight −1 if we change cartesian. In order to convey all this information, we represent weigths as linear combinations of the names of the bases:

```
In[232]:=
WeightOf[%]
```

```
Out[232]=
-cartesian + polar
```

If we worked with a single basis, a weight n *`basisname` could be understood as the traditional integer weight n

©2003–2004 José M. Martín–García

```
In[233]:=
WeightOf[etaUppolar[a, b, c] etaUppolar[d, e, f]]

Out[233]=
2 polar
```

The Jacobian is represented as

```
In[234]:=
Jacobian[cartesian, polar][]

    ** DefTensor: Defining Jacobiancartesianpolar[].

Out[234]=
J̃
```

It is a scalar with weight +1 with respect to cartesian and −1 with respect to polar, so it has two tildes of different colours.

```
In[235]:=
WeightOf[%]

Out[235]=
cartesian - polar

In[236]:=
Jacobian[cartesian, polar][] * Jacobian[polar, cartesian][]

Out[236]=
1
```

xCoba` knows how to compute derivatives of Jacobians

```
In[237]:=
PD[-a][Jacobian[cartesian, polar][]] // ScreenDollarIndices

Out[237]=
-Γ[𝒟, 𝒟]^b_ab J̃

In[238]:=
LieD[v[a]][Jacobian[cartesian, polar][]] // ScreenDollarIndices

Out[238]=
J̃ (-e_a^b (𝒟_b v^a) + e^a_c (𝒟_a v^c))

In[239]:=
ContractBasis[%, OverDerivatives → True] // Simplify

Out[239]=
J̃ (-(𝒟_b v^b) + 𝒟_c v^c)
```

4.2. Determinants

Det	Determinant of a tensor of arbitrary rank

Determinants

©2003–2004 José M. Martín–García

We can define the determinant of a tensor in any given basis through the η tensors:

```
In[240]:=
Det[T[a, b], polar]
```

$$\text{Out[240]=}\quad \frac{1}{6}\,\eta_{ace}\,\eta_{bdf}\,T^{ab}\,T^{cd}\,T^{ef}$$

```
In[241]:=
WeightOf[%]
```

```
Out[241]=
-2 polar
```

Notice how the determinant depends on the basis, unless there are the same number of contravariant and covariant indices.

```
In[242]:=
DefTensor[T2[a, -b], M3, PrintAs → "T"]

    ** DefTensor: Defining tensor T2[a, -b].
```

```
In[243]:=
Det[T2[a, -b], polar]
```

$$\text{Out[243]=}\quad \frac{1}{6}\,\delta^{bdf}{}_{ace}\,T^{a}{}_{b}\,T^{c}{}_{d}\,T^{e}{}_{f}$$

```
In[244]:=
WeightOf[%]
```

```
Out[244]=
0
```

If a tensor has an odd number of indices, its determinant is zero

```
In[245]:=
DefTensor[T3[a, b, c], M3, PrintAs → "T"]

    ** DefTensor: Defining tensor T3[a, b, c].
```

```
In[246]:=
Det[T3[a, b, c], cartesian]
```

$$\text{Out[246]=}\quad \frac{1}{6}\,\eta_{adf1}\,\eta_{bef2}\,\eta_{cff3}\,T^{abc}\,T^{def}\,T^{f1f2f3}$$

```
In[247]:=
ToCanonical[%]
```

```
Out[247]=
0
```

```
In[248]:=
DefTensor[T4[a, b, c, -d], M3, PrintAs → "T"]

    ** DefTensor: Defining tensor T4[a, b, c, -d].
```

©2003–2004 José M. Martín–García

```
In[249]:=
Det[T4[a, b, c, -d], cartesian] // ToCanonical
```

```
Out[249]=
```

$$\frac{1}{6}\,\underset{\sim}{\eta}_{\text{bff4}}\,\underset{\sim}{\eta}_{\text{cf1f5}}\,\delta_{\text{aef3}}{}^{\text{df2f6}}\,\text{T}^{\text{abc}}{}_{\text{d}}\,\text{T}^{\text{eff1}}{}_{\text{f2}}\,\text{T}^{\text{f3f4f5}}{}_{\text{f6}}$$

Even if we know that the determinant of a given object is going to be basis independent, we must provide some basis:

```
In[250]:=
{Det[delta[a, -b]], Det[delta[a, -b], polar]}
```

```
Out[250]=
{Det[δ^a_b], 1}
```

```
In[251]:=
UndefTensor /@ {T2, T3, T4};
```

```
** UndefTensor: Undefined tensor T2

** UndefTensor: Undefined tensor T3

** UndefTensor: Undefined tensor T4
```

4.3. The determinant of the metric and the ϵ tensor

AbsDet	Absolute value of the determinant of the metric in any given basis
epsilonToetaDown	Transform an ϵ tensor into an etaUp tensor
epsilonToetaUp	Transform an ϵ tensor into an etaDown tensor
etaDownToepsilon	Transform an etaDown tensor into an ϵ tensor
etaUpToepsilon	Transform an etaUp tensor into an ϵ tensor
$epsilonSign	Global sign of the ϵ tensor

Determinat of the metric; relation between η and ϵ tensors

The determinant of the metric is a weight 2 density

©2003–2004 José M. Martín–García

In[252]:=
```
DefMetric[-1, metric[-a, -b], cd, {"|", "D"}, PrintAs → "g"]
```

```
** DefTensor: Defining symmetric metric tensor metric[-a, -b].

** DefTensor: Defining antisymmetric tensor epsilonmetric[a, b, c].

** DefCovD: Defining covariant derivative cd[-a].

** DefTensor: Defining vanishing torsion tensor Torsioncd[a, -b, -c].

** DefTensor: Defining symmetric Christoffel tensor Christoffelcd[a, -b, -c].

** DefTensor: Defining Riemann tensor Riemanncd[-a, -b, -c, -d].

** DefTensor: Defining symmetric Ricci tensor Riccicd[-a, -b].

** DefCovD:  Contractions of Riemann automatically replaced by Ricci.

** DefTensor: Defining Ricci scalar RicciScalarcd[].

** DefCovD:  Contractions of Ricci automatically replaced by RicciScalar.

** DefTensor: Defining symmetric Einstein tensor Einsteincd[-a, -b].

** DefTensor: Defining vanishing Weyl tensor Weylcd[-a, -b, -c, -d].

** DefTensor: Defining symmetric TFRicci tensor TFRiccicd[-a, -b].

   Rules {1, 2} have been declared as DownValues for TFRiccicd.

** DefCovD:  Computing RiemannToWeylRules for dim 3

** DefCovD:  Computing RicciToTFRicci for dim 3

** DefCovD:  Computing RicciToEinsteinRules for dim 3
```

In[253]:=
```
AbsDet[metric, polar][]
```

```
** DefTensor: Defining weight +2 density
 AbsDetmetricpolar[]. Absolute value of determinant.
```

Out[253]=

$$|\tilde{g}|$$

Once we have a metric, we can build new tensors (with zero weight) from the etaUp and etaDown tensors. These new ϵ tensors depend only on the metric

In[254]:=
```
epsilonmetric[a, b, c]
```

Out[254]=

$$\in g^{abc}$$

In[255]:=
```
WeightOf[%]
```

Out[255]=

0

©2003–2004 José M. Martín–García

```
In[256]:=
etaUp[cartesian][a, b, c] /. etaUpToepsilon[cartesian, metric]

    ** DefTensor: Defining weight +2 density
     AbsDetmetriccartesian[]. Absolute value of determinant.

Out[256]=
-√|g̃| ϵg^abc

In[257]:=
% /. epsilonToetaDown[metric, cartesian]

Out[257]=
-|g̃| η̃^abc

In[258]:=
%% /. epsilonToetaUp[metric, cartesian]

Out[258]=
η̃^abc
```

The global sign for ϵ tensors is controlled by the global variable `$epsilonSign`

```
In[259]:=
? $epsilonSign

    $epsilonSign gives the sign of the epsilon tensor when all covariant
      components have been sorted. It is used in the relation between
      the epsilon tensor and the eta tensorial densities, in xCoba.
```

■ 5. Tracing contractions

One of the most important functions of `xCoba`` is `TraceBasisDummy`. We already know how to express a tensor on a given basis, as a contraction with `Basis` objects. With this new function, we can expand that contraction as a sum of coordinate components.

`TraceBasisDummy`	Expand dummy basis indices into their coordinate ranges

Tracing basis contractions.

```
In[260]:=
TraceBasisDummy[v[{a, cartesian}] v[{-a, -cartesian}]]

Out[260]=
v_0 v^0 + v_1 v^1 + v_2 v^2
```

A shorthand is provided. `v[{a,cart}]∘v[{a,cart}]`, where ∘ is `SmallCircle` (`⎵sc⎵`), has the same effect

```
In[261]:=
v[{a, cartesian}]∘v[{-a, -cartesian}]

Out[261]=
v_0 v^0 + v_1 v^1 + v_2 v^2
```

Again, there is complete control over which dummies should be expanded. The complete syntax is `TraceBasisDummy[expr, indices]`, where indices is one of the following:

©2003–2004 José M. Martín–García

– A single `BIndex`.

– A list of basis indices with head Indices.

– A vbundle (all basis dummies belonging to it will we expanded).

– A basis

```
In[262]:=
traceexpr = S[{-A, -comp}, {A, comp}]
  T[{-a, -polar}, {-b, -cartesian}] v[{a, polar}] v[{b, cartesian}]
```

```
Out[262]=
S_A^A T_ab v^a v^b
```

```
In[263]:=
TraceBasisDummy[traceexpr] // Simplify
```

```
Out[263]=
(S_-1^-1 + S_1^1)
 (T_00 v^0 v^0 + T_01 v^0 v^1 + T_10 v^0 v^1 + T_11 v^1 v^1 + T_02 v^0 v^2 + T_12 v^1 v^2 + T_20 v^0 v^2 + T_21 v^1 v^2 + T_22 v^2
```

```
In[264]:=
TraceBasisDummy[traceexpr, TangentM3] // Simplify
```

```
Out[264]=
S_A^A
 (T_00 v^0 v^0 + T_01 v^0 v^1 + T_10 v^0 v^1 + T_11 v^1 v^1 + T_02 v^0 v^2 + T_12 v^1 v^2 + T_20 v^0 v^2 + T_21 v^1 v^2 + T_22 v^2
```

```
In[265]:=
TraceBasisDummy[traceexpr, polar] // Simplify
```

```
Out[265]=
S_A^A (T_0b v^0 + T_1b v^1 + T_2b v^2) v^b
```

```
In[266]:=
TraceBasisDummy[traceexpr, cartesian] // Simplify
```

```
Out[266]=
S_A^A (T_a0 v^0 + T_a1 v^1 + T_a2 v^2) v^a
```

With derivatives,

```
In[267]:=
TraceBasisDummy[PD[{-b, -polar}][v[{b, polar}]]]
```

```
Out[267]=
∂_0 v^0 + ∂_1 v^1 + ∂_2 v^2
```

```
In[268]:=
(u[B] + S[{1, -comp}, {A, comp}] S[{-A, -comp}, B]) + PD[{A, comp}][S[{-A, -comp}, B]]
  (u[{C, comp}] + S[{-D, -comp}, {C, comp}] u[{D, comp}]) u[{-C, -comp}]
```

```
Out[268]=
S_1^A S_A^B + u^B + u_C (u^C + S_D^C u^D) ∂^A S_A^B
```

©2003–2004 José M. Martín–García

```
In[269]:=
TraceBasisDummy@%

Out[269]=
S_-1^B S_1^-1 + S_1^B S_1^1 + u^B + u_-1 (u^-1 + S_-1^-1 u^-1 + S_1^-1 u^1) (∂^-1 S_-1^B + ∂^1 S_1^B) +
 u_1 (S_-1^1 u^-1 + u^1 + S_1^1 u^1) (∂^-1 S_-1^B + ∂^1 S_1^B)

In[270]:=
Simplify[%]

Out[270]=
S_-1^B S_1^-1 + S_1^B S_1^1 + u^B + u_-1 ((1 + S_-1^-1) u^-1 + S_1^-1 u^1) (∂^-1 S_-1^B + ∂^1 S_1^B) +
 u_1 (S_-1^1 u^-1 + (1 + S_1^1) u^1) (∂^-1 S_-1^B + ∂^1 S_1^B)
```

Another example

```
In[271]:=
TraceBasisDummy[PDpolar[{-a, -cartesian}][v[{a, cartesian}]]]

Out[271]=
𝒟_0 v^0 + 𝒟_1 v^1 + 𝒟_2 v^2

In[272]:=
SeparateBasis[AIndex][%] // Expand

Out[272]=
e^b_0 Γ[𝒟, 𝒟]^0_ba v^a + e^b_1 Γ[𝒟, 𝒟]^1_ba v^a +
 e^b_2 Γ[𝒟, 𝒟]^2_ba v^a + e_a^0 e^b_0 (𝒟_b v^a) + e_a^1 e^b_1 (𝒟_b v^a) + e_a^2 e^b_2 (𝒟_b v^a)
```

■ 6. Working with components

We need some way to supply values for the components of a tensor and to extract components from abstract expressions. This section describes how to generate lists of components. We will assign them values in Section 7.

6.1. ComponentArray and TableOfComponents

`ComponentArray`	Generate a list of components
`TableOfComponents`	Generate a list of components, with many more options

Arrays of components

The commands described in this subsection essentially generate lists of c–indices from free b–indices (or free a–indices through `ToBasis`). The basic command is `ComponentArray`,

```
In[273]:=
ComponentArray[v[{a, polar}]]

Out[273]=
{v^0, v^1, v^2}
```

NOTE: Up to version 0.6, the role of `ComponentArray` was played by a different function, called `Components`.

If the object to expand is a tensor (and only in this case), we can use a shorter notation

©2003–2004 José M. Martín–García

```
In[274]:=
ComponentArray[T, {-polar, -polar}]

Out[274]=
{{T00, T01, T02}, {T10, T11, T12}, {T20, T21, T22}}
```

We can mix bases

```
In[275]:=
ComponentArray[T, {-polar, -cartesian}]

Out[275]=
{{T00, T01, T02}, {T10, T11, T12}, {T20, T21, T22}}
```

Component indices are not expanded

```
In[276]:=
ComponentArray[T[{-a, -polar}, {1, -polar}]]

Out[276]=
{T01, T11, T21}
```

We can specify which indices to expand

```
In[277]:=
ComponentArray[T[-{a, polar}, -{b, polar}] v[{c, cartesian}],
 IndexList[{-a, -polar}, {c, cartesian}]]

Out[277]=
{{T0b v0, T0b v1, T0b v2}, {T1b v0, T1b v1, T1b v2}, {T2b v0, T2b v1, T2b v2}}

In[278]:=
ComponentArray[T[-{a, polar}, -{b, polar}] v[{c, cartesian}], IndicesOf[T]]

Out[278]=
{{T00 vc, T01 vc, T02 vc}, {T10 vc, T11 vc, T12 vc}, {T20 vc, T21 vc, T22 vc}}
```

---

but c–indices are never expanded

```
In[279]:=
ComponentArray[v[{1, polar}], IndexList[{1, polar}]]

Out[279]=
v1
```

The order of indices matters:

```
In[280]:=
ComponentArray[v[{a, polar}, {b, polar}]]

Out[280]=
{{v00, v01, v02}, {v10, v11, v12}, {v20, v21, v22}}

In[281]:=
ComponentArray[v[{b, polar}, {a, polar}]]

Out[281]=
{{v00, v10, v20}, {v01, v11, v21}, {v02, v12, v22}}
```

©2003–2004 José M. Martín–García

`ComponentArray` avoids clashes with dummy indices (notice the use of Scalar):

```
In[282]:=
ComponentArray[Scalar[v[{a, polar}] v[{-a, -polar}]] v[{-a, -polar}]]

Out[282]=
{Scalar[v_a v^a] v_0, Scalar[v_a v^a] v_1, Scalar[v_a v^a] v_2}
```

If we want more flexibility, we can use `TableOfComponents`. This function, whose syntax mimics that of the standard *Mathematica* `Table`, is much more flexible, but also slower. The reason for this is that `TableOfComponents` uses `ToBasis` internally, which performs many checks. Now we specifiy 'iterators' on basis indices as additional arguments. If the specified index `{a, basis}` was already in the expression, this just tells the function when to expand it (i.e. in what order with respect to the remaining indices). If it was not present but `a` appeared as an abstract index, the system first turns it into a basis index and then expands it:

```
In[283]:=
TableOfComponents[T[-{a, polar}, -{b, polar}], -{a, polar}, -{b, polar}]

Out[283]=
{{T_00, T_01, T_02}, {T_10, T_11, T_12}, {T_20, T_21, T_22}}

In[284]:=
TableOfComponents[T[-a, -b], -{a, polar}, -{b, polar}]

Out[284]=
{{T_00, T_01, T_02}, {T_10, T_11, T_12}, {T_20, T_21, T_22}}
```

Notice how basis indices mimic the behaviour of *Mathematica* iterators. We have been able to expand an expression which contained only abstract indices. This is equivalent to doing

```
In[285]:=
ToBasis[polar][T[-a, -b]]

Out[285]=
T_ab

In[286]:=
ComponentArray[%]

Out[286]=
{{T_00, T_01, T_02}, {T_10, T_11, T_12}, {T_20, T_21, T_22}}
```

However, we cannot change the basis of a BIndex already in the expression

```
In[287]:=
TableOfComponents[T[{-a, -polar}, -b], {-a, -cartesian}]

Out[287]=
T_ab
```

If we do not specify any index, the argument is returned unchanged (compare with `ComponentArray`):

```
In[288]:=
TableOfComponents[v[{a, polar}] ]

Out[288]=
v^a
```

©2003–2004 José M. Martín–García

```
In[289]:=
ComponentArray[v[{a, polar}]]

Out[289]=
{v^0, v^1, v^2}
```

The order is again important

```
In[290]:=
TableOfComponents[T[a, b], {a, polar}, {b, polar}]

Out[290]=
{{T^00, T^01, T^02}, {T^10, T^11, T^12}, {T^20, T^21, T^22}}

In[291]:=
TableOfComponents[T[a, b], {b, polar}, {a, polar}]

Out[291]=
{{T^00, T^10, T^20}, {T^01, T^11, T^21}, {T^02, T^12, T^22}}
```

As `ToBasis` is able to handle derivatives, we can include them in the arguments of `TableOfComponents`

```
In[292]:=
TableOfComponents[PD[-a][v[b]], -{a, polar}, {b, cartesian}] // MatrixForm

Out[292]//MatrixForm=
```

$$\begin{pmatrix} -\Gamma[\mathcal{D}]^0_{0\,f\$1756}\, v^{f\$1756} + \partial_0 v^0 & -\Gamma[\mathcal{D}]^1_{0\,f\$1756}\, v^{f\$1756} + \partial_0 v^1 & -\Gamma[\mathcal{D}]^2_{0\,f\$1756}\, v^{f\$1756} + \partial_0 v^2 \\ -\Gamma[\mathcal{D}]^0_{1\,f\$1758}\, v^{f\$1758} + \partial_1 v^0 & -\Gamma[\mathcal{D}]^1_{1\,f\$1758}\, v^{f\$1758} + \partial_1 v^1 & -\Gamma[\mathcal{D}]^2_{1\,f\$1758}\, v^{f\$1758} + \partial_1 v^2 \\ -\Gamma[\mathcal{D}]^0_{2\,f\$1760}\, v^{f\$1760} + \partial_2 v^0 & -\Gamma[\mathcal{D}]^1_{2\,f\$1760}\, v^{f\$1760} + \partial_2 v^1 & -\Gamma[\mathcal{D}]^2_{2\,f\$1760}\, v^{f\$1760} + \partial_2 v^2 \end{pmatrix}$$

```
In[293]:=
TraceBasisDummy[%]

Out[293]=
{{-Γ[𝒟]^0_00 v^0 - Γ[𝒟]^0_01 v^1 - Γ[𝒟]^0_02 v^2 + ∂0 v^0,
  -Γ[𝒟]^1_00 v^0 - Γ[𝒟]^1_01 v^1 - Γ[𝒟]^1_02 v^2 + ∂0 v^1, -Γ[𝒟]^2_00 v^0 - Γ[𝒟]^2_01 v^1 - Γ[𝒟]^2_02 v^2 + ∂0 v^2},
 {-Γ[𝒟]^0_10 v^0 - Γ[𝒟]^0_11 v^1 - Γ[𝒟]^0_12 v^2 + ∂1 v^0, -Γ[𝒟]^1_10 v^0 - Γ[𝒟]^1_11 v^1 - Γ[𝒟]^1_12 v^2 + ∂1 v^1,
  -Γ[𝒟]^2_10 v^0 - Γ[𝒟]^2_11 v^1 - Γ[𝒟]^2_12 v^2 + ∂1 v^2}, {-Γ[𝒟]^0_20 v^0 - Γ[𝒟]^0_21 v^1 - Γ[𝒟]^0_22 v^2 + ∂2 v^0,
  -Γ[𝒟]^1_20 v^0 - Γ[𝒟]^1_21 v^1 - Γ[𝒟]^1_22 v^2 + ∂2 v^1, -Γ[𝒟]^2_20 v^0 - Γ[𝒟]^2_21 v^1 - Γ[𝒟]^2_22 v^2 + ∂2 v^2}}
```

If we wanted to obtain this result from $\partial_a\ v^b$ using `ComponentArray`, we would have to be much more careful.

Several vbundles

```
In[294]:=
TableOfComponents[v[a] u[A], {A, comp}, {a, cartesian}]

Out[294]=
{{u^-1 v^0, u^-1 v^1, u^-1 v^2}, {u^1 v^0, u^1 v^1, u^1 v^2}}
```

These functions do not take the symmetries into account automatically, but we can simplfy

©2003–2004 José M. Martín–García

```
In[295]:=
ComponentArray[T[{a, polar}, {b, polar}]] // ToCanonical // MatrixForm
```

```
Out[295]//MatrixForm=
```

$$\begin{pmatrix} T^{0\,0} & T^{0\,1} & T^{0\,2} \\ T^{0\,1} & T^{1\,1} & T^{1\,2} \\ T^{0\,2} & T^{1\,2} & T^{2\,2} \end{pmatrix}$$

Currently there is no function to construct a single component, because the notation must be nearly as long as the direct construction itself! `Coba` (see section 6.2) can be used to do that, in case the same component must be constructed for several different tensors.

6.2. ThreadComponentArray

6.3. Coba and CobaArray

`Coba`	Component of an arbitrary expression
`CobaArray`	Array of component of an arbitrary expression

Heads for components

`xCoba`` represents the concept of component of an arbitrary expression (a tensor, tensor product, covariant derivative, etc.) with the head `Coba`. Internally, `Coba` is treated as `IndexList`, but only allows bc–indices and is considered a tensor. It is output as a dotted box. This object has a more limited use than the function described above at user level, but it may be interesting for some sophisticated manipulations:

```
In[296]:=
Coba[{1, -cartesian}, {2, polar}, {1, -polar}]
```

```
Out[296]=
```

$$\square_{1}{}^{2}{}_{1}$$

```
In[297]:=
xTensorQ[Coba]
```

```
Out[297]=
True
```

The dotted box can stand for anything and there is no indication of which component of `Coba` goes to which slot of the expression in the box. However, the indices of `Coba` are sorted, so that `Coba[{1,cartesian}, {1,polar}] ≠ Coba[{1,polar}, {1,cartesian}]`.

```
In[298]:=
Coba[{1, -polar}, {2, -polar}] /. Coba → T
```

```
Out[298]=
```

$$T_{1\,2}$$

```
In[299]:=
Coba[{2, -polar}, {1, -polar}] /. Coba → T
```

```
Out[299]=
```

$$T_{2\,1}$$

©2003–2004 José M. Martín–García

```
In[300]:=
Coba[{1, -polar}, {2, -polar}] /. Coba → Function[T[#2, #1]]

Out[300]=
T2 1
```

There is a special notation for that:

```
In[301]:=
Coba[{1, -polar}, {2, -polar}][T]

Out[301]=
T1 2

In[302]:=
Coba[{1, -polar}, {2, -polar}][T[#2, #1]]

Out[302]=
T2 1

In[303]:=
Coba[{1, -polar}, {2, -polar}][PD[#1]@T[#2, #1] T[#1]]

Out[303]=
T1 ∂1 T2 1
```

The command `CobaArray` gives more flexibility and allows the user to generate all the components for given b–indices. An extended basis index notation is provided.

```
In[304]:=
CobaArray[{a, polar}]

Out[304]=
{□0, □1, □2}

In[305]:=
CobaArray[{a, polar}, {-A, -comp}]

Out[305]=
{{□0-1, □0 1}, {□1-1, □1 1}, {□2-1, □2 1}}
```

With component indices, `CobaArray` reduces to `Coba`,

```
In[306]:=
CobaArray[{1, polar}, {1, -comp}]

Out[306]=
□1 1
```

We can manually specify which components should be included as a third element in the {} of the basis index

```
In[307]:=
CobaArray[{a, polar, {1, 2}}]

Out[307]=
{□1, □2}
```

©2003–2004 José M. Martín–García

```
In[308]:=
CobaArray[{a, polar, {0, 0, 1, 1, 2, 2}}]

Out[308]=
{□^0, □^0, □^1, □^1, □^2, □^2}
```

We can get all the components of a tensor this way

```
In[309]:=
CobaArray[{a, polar}, {-b, -polar}, {-c, -polar}]

Out[309]=
{{{□^0_00, □^0_01, □^0_02}, {□^0_10, □^0_11, □^0_12}, {□^0_20, □^0_21, □^0_22}},
 {{□^1_00, □^1_01, □^1_02}, {□^1_10, □^1_11, □^1_12}, {□^1_20, □^1_21, □^1_22}},
 {{□^2_00, □^2_01, □^2_02}, {□^2_10, □^2_11, □^2_12}, {□^2_20, □^2_21, □^2_22}}}

In[310]:=
% /. Coba → Function[PD[#1]@T[#2, #3]] // MatrixForm

Out[310]//MatrixForm=
```

$$\begin{pmatrix} \begin{pmatrix} \partial^0 T_{0\,0} \\ \partial^0 T_{0\,1} \\ \partial^0 T_{0\,2} \end{pmatrix} & \begin{pmatrix} \partial^0 T_{1\,0} \\ \partial^0 T_{1\,1} \\ \partial^0 T_{1\,2} \end{pmatrix} & \begin{pmatrix} \partial^0 T_{2\,0} \\ \partial^0 T_{2\,1} \\ \partial^0 T_{2\,2} \end{pmatrix} \\ \begin{pmatrix} \partial^1 T_{0\,0} \\ \partial^1 T_{0\,1} \\ \partial^1 T_{0\,2} \end{pmatrix} & \begin{pmatrix} \partial^1 T_{1\,0} \\ \partial^1 T_{1\,1} \\ \partial^1 T_{1\,2} \end{pmatrix} & \begin{pmatrix} \partial^1 T_{2\,0} \\ \partial^1 T_{2\,1} \\ \partial^1 T_{2\,2} \end{pmatrix} \\ \begin{pmatrix} \partial^2 T_{0\,0} \\ \partial^2 T_{0\,1} \\ \partial^2 T_{0\,2} \end{pmatrix} & \begin{pmatrix} \partial^2 T_{1\,0} \\ \partial^2 T_{1\,1} \\ \partial^2 T_{1\,2} \end{pmatrix} & \begin{pmatrix} \partial^2 T_{2\,0} \\ \partial^2 T_{2\,1} \\ \partial^2 T_{2\,2} \end{pmatrix} \end{pmatrix}$$

©2003–2004 José M. Martín–García

■ 7. Assigning values to components

We need some convenient way to store all the values of the components of a tensor (or, more generally, an arbitrary expression). `xCoba`` provides several tools to accomplish this, optimised to make full use of the symmetries. This section describes how to store and use these values, but not how to compute them. The latter is the object of Section 9 (not yet fully implemented).

Given a tensor there are several things we need to worry about when working with values:

– 1. Are we giving values to the components of the tensor itself or to some derivative of it?

– 2. The slot–symmetry of the expression.

– 3. What is the character of the c–indices involved?

– 4. Which bases are we using to form the component?

– 5. When did we construct a particular set of components?

– 6. Have we already computed values for all components of the expression?

This section describes a series of functions for the storage of tensor values. After reading it, we think it is very profitable to take a look at the notebook Schwarzschild.nb, by Alfonso García–Parrado, which describes an example computation with the Schwarzschild metric, employing some of the commands described here. This notebook is included in xCoba.tar.gz and can be downloaded from http://metric.iem.csic.es/Martin–Garcia/xAct/index.html.

7.1. FoldedRule and ValID

`FoldedRule`	List of lists of rules to be applied sequentially
`ValID`	Identifier for each independent set of components stored for a tensor
`TensorValIDs`	List of all the ValIDs of a given tensor
`DateOfValID`	Time a given ValID was generated

Identifying sets of component values

Before introducing the general functions, let us consider a simple example. Let us suppose we have a rank two tensor U_{ab}, with no symmetries and only one basis. Then, the straightforward way of storing its components would be with a list of replacement rules:

$$\{U_{11} \to a,\ U_{12} \to b,\ U_{13} \to c,\ \ldots\}$$

But now consider, still with just one basis in play and no metric, that our tensor T_{ab} is symmetric. Then it would be a waste to store twice the component $T_{12} = T_{21}$, for example. With higher rank tensors and higher symmetries the redundancy makes the process very inefficient. Instead, we could think of storing components as a pair of lists. The first one would give the relations due to symmetry and the second one the values of the independent components. We could have something like

$$\{\{T_{21} \to T_{12},\ T_{31} \to T_{13},\ T_{32} \to T_{23}\},\ \{T_{11} \to a,\ T_{12} \to b,\ T_{13} \to c,\ T_{22} \to d,\ T_{23} \to e, T_{33} \to f\}\}$$

It may appear we have gained nothing, after all we still have 9 rules. However, all the information of the first list is determined by the symmetry group of the tensor, which `xCoba`` already knows since its definition. Also, the independent values may be very large expressions while the dependent rules are always very simple. This structure is represented in `xCoba`` with the concept of FoldedRule (defined in the auxiliary package `xCore``):

©2003–2004 José M. Martín–García

```
In[311]:=
?FoldedRule

    FoldedRule[rules1, rules2, ...] contains a number of lists
      of rules which are applied sequentially (first rules1, then
      rules2, etc.) when called by ReplaceAll and ReplaceRepeated.
```

Example:

```
In[312]:=
x + t /. FoldedRule[{x → y, t → x}, {y → z}]

Out[312]=
x + z
```

In the next subsection we shall see how `xCoba`` generates these lists. The next problem comes when we consider a more general case where we have a metric and several bases. In this situation we may want to store the values for a given tensor in different bases and with different index characters. However, not all those combinations are independent. Recalling our symmetric tensor T_{ab}, we see that the FoldedRules that would contain its values for $T_a{}^b$ and $T^b{}_a$ would have the same independent components (likewise for T_{ab} and T_{ba}). So among all sets of values for a tensor, only some will be independent. `xCoba`` identifies these independent sets with the concept of `ValID`:

```
In[313]:=
?ValID

    ValID[tensor, {{basis1, basis2, ...}}] identifies a set of values for the
      tensor with c-indices respectively in basis1, basis2, ... (with sign
      indicating character as usual; LI and -LI are accepted). For a tensor
      with symmetry, ValID[tensor, {bases1, bases2, ...}] identifies a set
      of values related by that symmetry, where each of the basesi is a list
      of possible reorderings of the bases of the c-indices. ValID[tensor,
      der1, der2, ..., bases] identifies a set of values for the given
      derivatives of the tensor in Postfix order (in Prefix notation it would
      mean ...[der2[der1[tensor]]), where bases is a list of lists as before.
```

Continuing with our example, we would have for our tensor U_{ab} a single ValID, which would be

```
ValID[U, { {polar, polar}} ]
```

(both indices in the base polar). We have several for T_{ab}

```
ValID[T, { {polar, -polar} , {-polar, polar}} ]

ValID[T, { {polar, cartesian}, {cartesian, polar}]
```

The first would represent the fact that $T_a{}^b$ and $T^b{}_a$ have the same independent rules and the second would represent the situation for T_{ab} and T_{ba}. Notice how even in the case of U_{ab}, where there is only one possible basis configuration in the `ValID`, we give a list of lists of bases. The command `VTensorValIDs` returns all the `ValIDs` we have stored for a given tensors (and its derivatives, as we shall see below).

All of this will be much clearer once we see how `xCoba`` generates the FoldedRules and their corresponding `ValIDs` for a tensor in the next subsection.

©2003–2004 José M. Martín–García

7.2. ComponentValue and TensorValues without independent values

`ComponentValue`	Give the value of a single component
`TensorValues`	List of all values for a given tensor
`TensorValIDs`	List of all the ValIDs of a given tensor
`DateOfValID`	Time a given ValID was generated
`$CVVerbose`	Switch verbose output on/off for `ComponentValue`

Storing values for components

We have seen how `xCoba`` stores and labels sets of tensor values for different basis configurations; this section describes how these sets are generated. The basic function is `ComponentValue`, which generates the rule for a single component. `ComponentValue` can be called with one or two arguments:

```
In[314]:=
?ComponentValue

    ComponentValue[expr], for a component tensor expression (a tensor or a
      derivative of a tensor), returns a value rule of the form expr -> canon
      where canon is the canonical form of expr under ToCanonical. This value rule
      is automatically stored in the TensorValues list of that tensor so that it
      can be used from that list without having to recompute it. ComponentValue[
      expr, value] returns a value rule expr -> value, and stores internally both
      the dependent rule expr -> canon and the independent rule canon -> value.
```

We begin by examining how it works with just one argument (i.e. without specifying an independent value). Let us define a totally antisymmetric tensor

```
In[315]:=
DefTensor[W[a, b, c], M3, Antisymmetric[{a, b, c}]]

    ** DefTensor: Defining tensor W[a, b, c].

In[316]:=
SetOptions[CanonicalPerm, MathLink → True]

Out[316]=
{MathLink → True, TimeVerbose → False, xPermVerbose → False, OrderedBase → True}
```

Suppose we want to generate a rule for one of its components

```
In[317]:=
W[{0, polar}, {1, polar}, {2, polar}]

Out[317]=
W^{0 1 2}

In[318]:=
ComponentValue[%]

    Added independent rule W^{0 1 2} → W^{0 1 2} for tensor W

Out[318]=
W^{0 1 2} → W^{0 1 2}
```

©2003–2004 José M. Martín–García

```
In[319]:=
TensorValues[W]

Out[319]=
FoldedRule[{}, {W^012 → W^012}]
```

As we can see, `xCoba`` has recognised that W^{012} is an independent component and has added a rule for it in the second list of the FoldedRule. This rule is trivial, as we have not specified the numerical value of the component. But consider now the following

```
In[320]:=
ComponentValue[W[{2, polar}, {1, polar}, {0, polar}]]

    Added dependent rule W^210 → -W^012 for tensor W

Out[320]=
W^210 → -W^012

In[321]:=
TensorValues[W]

Out[321]=
FoldedRule[{W^210 → -W^012}, {W^012 → W^012}]
```

`xCoba`` sees that W^{210} is not an independent component and generates for it a rule in the first list of the FoldedRule. We can easily generate the full set of component values:

```
In[322]:=
W[{a, polar}, {b, polar}, {c, polar}] // ComponentArray // Flatten

Out[322]=
{W^000, W^001, W^002, W^010, W^011, W^012, W^020, W^021, W^022, W^100, W^101, W^102, W^110,
 W^111, W^112, W^120, W^121, W^122, W^200, W^201, W^202, W^210, W^211, W^212, W^220, W^221, W^222}
```

©2003–2004 José M. Martín–García

```
In[323]:=
ComponentValue /@ %

Added dependent rule W^000 → 0 for tensor W
Added dependent rule W^001 → 0 for tensor W
Added dependent rule W^002 → 0 for tensor W
Added dependent rule W^010 → 0 for tensor W
Added dependent rule W^011 → 0 for tensor W
Added dependent rule W^020 → 0 for tensor W
Added dependent rule W^021 → -W^012 for tensor W
Added dependent rule W^022 → 0 for tensor W
Added dependent rule W^100 → 0 for tensor W
Added dependent rule W^101 → 0 for tensor W
Added dependent rule W^102 → -W^012 for tensor W
Added dependent rule W^110 → 0 for tensor W
Added dependent rule W^111 → 0 for tensor W
Added dependent rule W^112 → 0 for tensor W
Added dependent rule W^120 → W^012 for tensor W
Added dependent rule W^121 → 0 for tensor W
Added dependent rule W^122 → 0 for tensor W
Added dependent rule W^200 → 0 for tensor W
Added dependent rule W^201 → W^012 for tensor W
Added dependent rule W^202 → 0 for tensor W
Added dependent rule W^211 → 0 for tensor W
Added dependent rule W^212 → 0 for tensor W
Added dependent rule W^220 → 0 for tensor W
Added dependent rule W^221 → 0 for tensor W
Added dependent rule W^222 → 0 for tensor W

Out[323]=
{W^000 → 0, W^001 → 0, W^002 → 0, W^010 → 0, W^011 → 0, W^012 → W^012, W^020 → 0, W^021 → -W^012, W^022 →
 W^100 → 0, W^101 → 0, W^102 → -W^012, W^110 → 0, W^111 → 0, W^112 → 0, W^120 → W^012, W^121 → 0, W^122 →
 W^200 → 0, W^201 → W^012, W^202 → 0, W^210 → -W^012, W^211 → 0, W^212 → 0, W^220 → 0, W^221 → 0, W^222 →
```

As we can see by reading the messages, the rules for W^{012} and W^{210} have not been recomputed. Now

©2003–2004 José M. Martín–García

```
In[324]:=
TensorValues[W]

Out[324]=
FoldedRule[{W^210 → -W^012, W^000 → 0, W^001 → 0, W^002 → 0, W^010 → 0, W^011 → 0,
  W^020 → 0, W^021 → -W^012, W^022 → 0, W^100 → 0, W^101 → 0, W^102 → -W^012, W^110 → 0,
  W^111 → 0, W^112 → 0, W^120 → W^012, W^121 → 0, W^122 → 0, W^200 → 0, W^201 → W^012,
  W^202 → 0, W^211 → 0, W^212 → 0, W^220 → 0, W^221 → 0, W^222 → 0}, {W^012 → W^012}]
```

There is only one independent rule. We have one `ValID` for W^{abc}, which consists of only one basis configuration

```
In[325]:=
TensorValIDs[W]

Out[325]=
{ValID[W, {{polar, polar, polar}}]}
```

`ValIDs` are timestamped:

```
In[326]:=
DateOfValID /@ %

Out[326]=
{{2008, 5, 14, 18, 56, 22.950787}}
```

Let us now consider an example with several bases. Recall that T_{ab} is a symmetric tensor

```
In[327]:=
ComponentValue[T[{2, -polar}, {1, -cartesian}]]

    Added dependent rule T_21 → T_12 for tensor T

Out[327]=
T_21 → T_12

In[328]:=
T[-{a, polar}, -{b, cartesian}] // ComponentArray // Flatten

Out[328]=
{T_00, T_01, T_02, T_10, T_11, T_12, T_20, T_21, T_22}
```

©2003–2004 José M. Martín–García

```
In[329]:=
ComponentValue /@ %

    Added dependent rule T00 → T00 for tensor T

    Added independent rule T01 → T01 for tensor T

    Added independent rule T02 → T02 for tensor T

    Added dependent rule T10 → T01 for tensor T

    Added dependent rule T11 → T11 for tensor T

    Added independent rule T12 → T12 for tensor T

    Added dependent rule T20 → T02 for tensor T

    Added dependent rule T22 → T22 for tensor T

Out[329]=
{T00 → T00, T01 → T01, T02 → T02, T10 → T01, T11 → T11, T12 → T12, T20 → T02, T21 → T12, T22 →

In[330]:=
TensorValIDs[T]

Out[330]=
{ValID[T, {{-cartesian, -polar}, {-polar, -cartesian}}]}
```

We still do not have all the rules, because our ValID mixes the cases T_{ab} and T_{ba}:

```
In[331]:=
T[-{a, cartesian}, -{b, polar}] // ComponentArray // Flatten

Out[331]=
{T00, T01, T02, T10, T11, T12, T20, T21, T22}

In[332]:=
ComponentValue /@ %

    Added independent rule T00 → T00 for tensor T

    Added independent rule T01 → T01 for tensor T

    Added independent rule T02 → T02 for tensor T

    Added dependent rule T10 → T01 for tensor T

    Added independent rule T11 → T11 for tensor T

    Added independent rule T12 → T12 for tensor T

    Added dependent rule T20 → T02 for tensor T

    Added dependent rule T21 → T12 for tensor T

    Added independent rule T22 → T22 for tensor T

Out[332]=
{T00 → T00, T01 → T01, T02 → T02, T10 → T01, T11 → T11, T12 → T12, T20 → T02, T21 → T12, T22 →
```

©2003–2004 José M. Martín–García

```
In[333]:=
TensorValues[T]

Out[333]=
FoldedRule[
 {T21 → T12, T00 → T00, T10 → T01, T11 → T11, T20 → T02, T22 → T22, T10 → T01, T20 → T02, T21 -
 {T01 → T01, T02 → T02, T12 → T12, T00 → T00, T01 → T01, T02 → T02, T11 → T11, T12 → T12, T22 -

In[334]:=
TensorValIDs[T]

Out[334]=
{ValID[T, {{-cartesian, -polar}, {-polar, -cartesian}}]}
```

Now we still only have one `ValID` for T_{ab}, but it consists of two basis configurations. As we can see, the FoldedRule returned by `TensorValues` mixes the cases T_{ab} and T_{ba}. Another example, now mixing index characters:

```
In[335]:=
ComponentArray[T[{a, polar}, -{b, cartesian}]] // Flatten

Out[335]=
{T0 0, T0 1, T0 2, T1 0, T1 1, T1 2, T2 0, T2 1, T2 2}

In[336]:=
ComponentValue /@ %

    Added dependent rule T0 0 → T0 0 for tensor T

    Added independent rule T0 1 → T0 1 for tensor T

    Added independent rule T0 2 → T0 2 for tensor T

    Added dependent rule T1 0 → T0 1 for tensor T

    Added dependent rule T1 1 → T1 1 for tensor T

    Added independent rule T1 2 → T1 2 for tensor T

    Added dependent rule T2 0 → T0 2 for tensor T

    Added dependent rule T2 1 → T1 2 for tensor T

    Added dependent rule T2 2 → T2 2 for tensor T

Out[336]=
{T0 0 → T0 0, T0 1 → T0 1, T0 2 → T0 2, T1 0 → T0 1, T1 1 → T1 1, T1 2 → T1 2, T2 0 → T0 2, T2 1 → T1 2, T2 2 → T2 2}

In[337]:=
ComponentArray[T[{-a, cartesian}, {a, polar}]] // Flatten

Out[337]=
{T0 0, T0 1, T0 2, T1 0, T1 1, T1 2, T2 0, T2 1, T2 2}
```

©2003–2004 José M. Martín–García

In[338]:=
```
ComponentValue /@ %
```

```
Added independent rule T_0^0 → T_0^0 for tensor T

Added independent rule T_0^1 → T_0^1 for tensor T

Added independent rule T_0^2 → T_0^2 for tensor T

Added dependent rule T_1^0 → T^0_1 for tensor T

Added independent rule T_1^1 → T_1^1 for tensor T

Added independent rule T_1^2 → T_1^2 for tensor T

Added dependent rule T_2^0 → T^0_2 for tensor T

Added dependent rule T_2^1 → T^1_2 for tensor T

Added independent rule T_2^2 → T_2^2 for tensor T
```

Out[338]=
```
{T_0^0 → T_0^0, T_0^1 → T_0^1, T_0^2 → T_0^2, T_1^0 → T^0_1, T_1^1 → T_1^1, T_1^2 → T_1^2, T_2^0 → T^0_2, T_2^1 → T^1_2, T_2^2 → T_2^2}
```

In[339]:=
```
TensorValIDs[T]
```

Out[339]=
```
{ValID[T, {{-cartesian, polar}, {polar, -cartesian}}],
 ValID[T, {{-cartesian, -polar}, {-polar, -cartesian}}]}
```

We have two `ValIDs`, both consisting of two basis configurations. Notice how newer `ValIDs` appear first.We can retrieve the `TensorValues` for either one of them

In[340]:=
```
TensorValues[T, {{-cartesian, polar}, {polar, -cartesian}}]
```

Out[340]=
```
FoldedRule[
 {T^0_0 → T_0^0, T^1_0 → T_0^1, T^1_1 → T_1^1, T^2_0 → T_0^2, T^2_1 → T_1^2, T^2_2 → T_2^2, T_1^0 → T^0_1, T_2^0 → T^0_2, T_2^1 → T^1_2},
 {T^0_1 → T^0_1, T^0_2 → T^0_2, T^1_2 → T^1_2, T_0^0 → T_0^0, T_0^1 → T_0^1, T_0^2 → T_0^2, T_1^1 → T_1^1, T_1^2 → T_1^2, T_2^2 → T_2^2}]
```

or for all at the same time:

In[341]:=
```
TensorValues[T]
```

Out[341]=
```
FoldedRule[
 {T^0_0 → T_0^0, T^1_0 → T_0^1, T^1_1 → T_1^1, T^2_0 → T_0^2, T^2_1 → T_1^2, T^2_2 → T_2^2, T_1^0 → T^0_1, T_2^0 → T^0_2, T_2^1 → T^1_2},
 {T^0_1 → T^0_1, T^0_2 → T^0_2, T^1_2 → T^1_2, T_0^0 → T_0^0, T_0^1 → T_0^1, T_0^2 → T_0^2, T_1^1 → T_1^1, T_1^2 → T_1^2, T_2^2 → T_2^2},
 {T_2 1 → T_1 2, T_0 0 → T_0 0, T_1 0 → T_0 1, T_1 1 → T_1 1, T_2 0 → T_0 2, T_2 2 → T_2 2, T_1 0 → T_0 1, T_2 0 → T_0 2, T_2 1 -
 {T_0 1 → T_0 1, T_0 2 → T_0 2, T_1 2 → T_1 2, T_0 0 → T_0 0, T_0 1 → T_0 1, T_0 2 → T_0 2, T_1 1 → T_1 1, T_1 2 → T_1 2, T_2 2 -
```

We can delete single rules

©2003–2004 José M. Martín–García

```
In[342]:=
ComponentValue[T[{1, -cartesian}, {2, -polar}], Null]

    Dropped independent rule T12 → T12 for tensor T

Out[342]=
T12 → Null
```

We can also easily delete all `TensorValues` for a given tensor

```
In[343]:=
DeleteTensorValues[W]

    Deleted values for tensor W, derivatives {} and bases {{polar, polar, polar}}.
```

or only those for a single `ValID`

```
In[344]:=
DeleteTensorValues[T, {{-cartesian, polar}, {polar, -cartesian}}]

    Deleted values for tensor T, derivatives {}
      and bases {{-cartesian, polar}, {polar, -cartesian}}.
```

7.3. ComponentValue and TensorValues with independent values

Until now, the second list of the FoldedRules contained only trivial replacement rules such as $T_{11} \to T_{11}$. It is very simple to specify values for the LHS, we just have to call `ComponentValue` with two arguments:

```
In[345]:=
ComponentValue[T[{1, -polar}, {2, -polar}], 5]

    Added independent rule T12 → 5 for tensor T

Out[345]=
T12 → 5

In[346]:=
TensorValues[T, {{-polar, -polar}}]

Out[346]=
FoldedRule[{}, {T12 → 5}]
```

Notice what happens if we specify a dependent component

```
In[347]:=
ComponentValue[T[{1, -polar}, {0, -polar}], 6]

    Added dependent rule T10 → T01 for tensor T

    Added independent rule T01 → 6 for tensor T

Out[347]=
T10 → 6
```

First the dependent rule is added and then the value (with a possible sign) is assigned to the corresponding independent component:

©2003–2004 José M. Martín–García

```
In[348]:=
TensorValues[T, {{-polar, -polar}}]

Out[348]=
FoldedRule[{T10 → T01}, {T12 → 5, T01 → 6}]
```

`ComponentValue` does not allow the user to introduce an inconsistent value

```
In[349]:=
ComponentValue[W[{1, polar}, {1, polar}, {1, polar}], 5]

    Added dependent rule W111 → 0 for tensor W

Out[349]=
W111 → 0
```

We switch the rule generation messages off

```
In[350]:=
$CVVerbose = False;
```

`ComponentValue` is threaded on pairs of lists when it has two arguments. This allows us to generate all independent rules with just one command.

```
In[351]:=
values = Table[i + j, {i, 1, 3}, {j, 1, 3}]

Out[351]=
{{2, 3, 4}, {3, 4, 5}, {4, 5, 6}}

In[352]:=
ComponentValue[ComponentArray[T[{a, polar}, {b, polar}]], values]

Out[352]=
{{T00 → 2, T01 → 3, T02 → 4}, {T10 → 3, T11 → 4, T12 → 5}, {T20 → 4, T21 → 5, T22 → 6}}

In[353]:=
ColumnForm /@ TensorValues[T, {{polar, polar}}]

Out[353]=
FoldedRule[ T10 → T01 , T00 → 2 ]
           T20 → T02   T01 → 3
           T21 → T12   T02 → 4
                       T11 → 4
                       T12 → 5
                       T22 → 6
```

7.4. AllComponentValues

`AllComponentValues`	Generation of the list of all the rules for a given ValID, taking the different configu– rations into account

Generating all value rules

We have just described an easy way to generate all the rules for a `ValID` which consisted on only one index configura– tion. But as we saw before, a single `ComponentArray` will not generate all the independent rules for a `ValID` with

©2003–2004 José M. Martín–García

several index configurations. One such case was the `ValID` for T_{ab} and T_{ba}. The following command (which only works for tensors and not for derivatives as we shall see in Section 7.4.) applies `ComponentArray` as many times as necessary in order to generate all the rules.

```
In[354]:=
values = Table[Random[Integer, {0, 5}], {3}, {3}]

Out[354]=
{{0, 2, 3}, {4, 3, 0}, {0, 2, 1}}

In[355]:=
ColumnForm /@ AllComponentValues[T[-{a, polar}, -{b, cartesian}], values]

Out[355]=
FoldedRule[ T21 → T12 , T01 → 2 ]
           T00 → T00   T02 → 3
           T10 → T01   T12 → 0
           T11 → T11   T00 → 0
           T20 → T02   T01 → 4
           T22 → T22   T02 → 0
           T10 → T01   T11 → 3
           T20 → T02   T22 → 1
           T21 → T12   T12 → 2
```

We turn the messages back on:

```
In[356]:=
$CVVerbose = True;
```

7.5. Components and derivatives

`ComponentValue` also works with derivatives of tensors (but not for more general expressions, you can use `Coba` as a placeholder and then substitute the expression). Remember that with our definitions the components of the derivative are not the derivatives of the components.

```
In[357]:=
PD[{2, -cartesian}][v[{1, cartesian}]]

Out[357]=
∂2 v1

In[358]:=
ComponentValue[%]

    Added independent rule ∂2 v1 → ∂2 v1 for tensor v

Out[358]=
∂2 v1 → ∂2 v1

In[359]:=
PD[{1, -polar}][PDpolar[{2, -cartesian}][v[{1, cartesian}]]]

Out[359]=
∂1 𝒟2 v1
```

©2003–2004 José M. Martín–García

```
In[360]:=
ComponentValue[%]

    Added independent rule ∂1 𝒟2 v^1 → ∂1 𝒟2 v^1 for tensor v

Out[360]=
∂1 𝒟2 v^1 → ∂1 𝒟2 v^1
```

Derivatives are represented by additional central arguments in `ValID`:

```
In[361]:=
TensorValIDs[v] // ColumnForm

Out[361]=
ValID[v, PDpolar[-cartesian], PD[-polar], {{cartesian, -cartesian, -polar}}]
ValID[v, PD[-cartesian], {{cartesian, -cartesian}}]

In[362]:=
DeleteTensorValues[v]

    Deleted values for tensor v, derivatives
     {PDpolar[-cartesian], PD[-polar]} and bases {{cartesian, -cartesian, -polar}}.

    Deleted values for tensor v, derivatives
     {PD[-cartesian]} and bases {{cartesian, -cartesian}}.
```

7.6. BasisValues

`BasisValues`	Values for the components of basis changes

Components of the basis changes

The components of the basis change matrices require separate treatment and are represented by the `BasisValues` head. Here we do not allow derivatives, because the derivatives of these objects are automatically replaced by Christof–fel tensors.

```
In[363]:=
?BasisValues

    BasisValues[basis1, -basis2] and BasisValues[-basis2, basis1] give
      the list of component values for the matrix of change between those
      two bases. There is no automatic computation of the inverse matrix.

In[364]:=
$CVVerbose = True;

In[365]:=
ComponentValue[Basis[{1, polar}, {2, -cartesian}]]

    Added independent rule e^1_2 → e^1_2 for tensor Basis

Out[365]=
e^1_2 → e^1_2
```

©2003–2004 José M. Martín–García

```
In[366]:=
AllComponentValues[Basis[{a, polar}, {-b, -cartesian}], Table[Random[], {3}, {3}]]

    Added independent rule e_0^0 → 0.6191 for tensor Basis

    Added independent rule e^0_1 → 0.0145658 for tensor Basis

    Added independent rule e^0_2 → 0.293863 for tensor Basis

    Added independent rule e_0^1 → 0.499612 for tensor Basis

    Added independent rule e_1^1 → 0.584013 for tensor Basis

    Added independent value e^1_2 → 0.0644212 for tensor Basis

    Added independent rule e_0^2 → 0.172212 for tensor Basis

    Added independent rule e_1^2 → 0.813358 for tensor Basis

    Added independent rule e_2^2 → 0.635848 for tensor Basis

Out[366]=
    FoldedRule[{}, {e^1_2 → 0.0644212, e_0^0 → 0.6191, e^0_1 → 0.0145658, e^0_2 → 0.293863,
      e_0^1 → 0.499612, e_1^1 → 0.584013, e_0^2 → 0.172212, e_1^2 → 0.813358, e_2^2 → 0.635848}]

In[367]:=
    BasisValues[polar, -cartesian]

Out[367]=
    FoldedRule[{}, {e^1_2 → 0.0644212, e_0^0 → 0.6191, e^0_1 → 0.0145658, e^0_2 → 0.293863,
      e_0^1 → 0.499612, e_1^1 → 0.584013, e_0^2 → 0.172212, e_1^2 → 0.813358, e_2^2 → 0.635848}]
```

The inverse basis change is not automatically computed, because it may take a very long time

```
In[368]:=
    BasisValues[cartesian, -polar]

Out[368]=
    FoldedRule[{}, {}]
```

7.7. Components of the metric, MetricInBasis

MetricInBasis	Values for the components of the metric

Components of the metric

It is very common in General Relativity to start working from a known line element. Inded, the typical approach is to start from the components of a metric in a coordinated basis, which determine both the metric and the coordinate system unambiguously. xCoba‘ provides a specialised function to define the values for the components of a metric, which can be used on its own or as an option for DefBasis.

©2003–2004 José M. Martín–García

In[369]:=
```
? MetricInBasis
```

```
MetricInBasis[metric, -basis, matrix] stores the values in matrix for the
  covariant components of metric in the given basis. MetricInBasis[metric,
  -basis, diagonal] stores the given values for the diagonal components,
  and zero everywhere else. MetricInBasis[metric, -basis, "Orthonormal"]
  takes diagonal values from SignatureOfMetric[metric] and zero everywhere
  else. MetricInBasis[metric, -basis, "Orthogonal"] stores zero off-diagonal
  values, but does not define values on the diagonal. MetricInBasis[
  metric, basis, values] stores values for the contravariant components of
  the inverse metric, where values is any of the previous possibilities.
```

In[370]:=
```
$Metrics
```

Out[370]=
```
{metric}
```

In[371]:=
```
$CVVerbose = False;
```

In[372]:=
```
MetricInBasis[metric, -polar, Table[i + j, {i, 3}, {j, 3}]]
```

Out[372]=
$$\{\{g_{00} \to 2, g_{01} \to 3, g_{02} \to 4\}, \{g_{10} \to 3, g_{11} \to 4, g_{12} \to 5\}, \{g_{20} \to 4, g_{21} \to 5, g_{22} \to 6\}\}$$

In[373]:=
```
TensorValues[metric]
```

Out[373]=
$$\text{FoldedRule}[\{g_{10} \to g_{01}, g_{20} \to g_{02}, g_{21} \to g_{12}\}, \{g_{00} \to 2, g_{01} \to 3, g_{02} \to 4, g_{11} \to 4, g_{12} \to 5, g_{22} \to 6\}]$$

An example with a diagonal metric

In[374]:=
```
DeleteTensorValues[metric]
```

```
Deleted values for tensor metric, derivatives {} and bases {{-polar, -polar}}.
```

In[375]:=
```
MetricInBasis[metric, -polar, {1, 2, 3}]
```

Out[375]=
$$\{\{g_{00} \to 1, g_{01} \to 0, g_{02} \to 0\}, \{g_{10} \to 0, g_{11} \to 2, g_{12} \to 0\}, \{g_{20} \to 0, g_{21} \to 0, g_{22} \to 3\}\}$$

And know with an orthonormal one

In[376]:=
```
? SignatureOfMetric
```

```
SignatureOfMetric[metric] gives the signature of the metric, in
  the form of a list of three elements: {p1s, m1s, zeros} giving the
  numbers of +1's, -1's and zeros, respectively, always in this order.
```

In[377]:=
```
SignatureOfMetric[metric] ^= {2, 1, 0}
```

Out[377]=
```
{2, 1, 0}
```

©2003–2004 José M. Martín–García

```
In[378]:=
MetricInBasis[metric, -cartesian, "OrthoNormal"]

Out[378]=
{{g00 → 1, g01 → 0, g02 → 0}, {g10 → 0, g11 → 1, g12 → 0}, {g20 → 0, g21 → 0, g22 → -1}}

In[379]:=
TensorValues[metric]

Out[379]=
FoldedRule[{g10 → g01, g20 → g02, g21 → g12},
 {g00 → 1, g01 → 0, g02 → 0, g11 → 1, g12 → 0, g22 → -1},
 {g10 → g01, g20 → g02, g21 → g12}, {g00 → 1, g01 → 0, g02 → 0, g11 → 2, g12 → 0, g22 → 3}]
```

7.7. Changing bases or index characters

ChangeComponents	Compute new TensorValues from an already stored index configuration

Changing bases or index characters

Suppose we already have the tensor values for a given ValID and we want to generate the corresponding set for an (inequivalent) index configuration. We may want to perform a typical change of basis like T_{ab} to T_{ab}, for example. Or we may want to lower or raise some indices. Given starting and finishing index configurations, xCoba` can compute the best route between them with the function ChangeComponents

A simple example

```
In[380]:=
ChangeComponents[v[{a, polar}], v[-{a, polar}]]

    Computed v^a → g^a b v_b  in 0.010123 Seconds

Out[380]=
FoldedRule[{},
 {v^0 → g^00 v_0 + g^01 v_1 + g^02 v_2, v^1 → g^10 v_0 + g^11 v_1 + g^12 v_2, v^2 → g^20 v_0 + g^21 v_1 + g^22 v_2}]

In[381]:=
TensorValIDs[v]

Out[381]=
{ValID[v, {{polar}}], ValID[v, {{-polar}}]}

In[382]:=
DeleteTensorValues[v]

    Deleted values for tensor v, derivatives {} and bases {{polar}}.

    Deleted values for tensor v, derivatives {} and bases {{-polar}}.
```

A more complicated example, with an antisymmetric tensor Y_{ab}. To keep it shorter we can change the cnumbers of polar to make xCoba` think the manifold is two dimensional

```
In[383]:=
DefTensor[Y[a, b], M3, Antisymmetric[{1, 2}]]

    ** DefTensor: Defining tensor Y[a, b].
```

©2003–2004 José M. Martín–García

```
In[384]:=
CNumbersOf[polar] ^= {1, 2}

Out[384]=
{1, 2}
```

We set some configuration global variables

```
In[385]:=
$CVVerbose = True;
$CVReplace = False;
$CCSimplify = Expand;
$UseValues = ToCanonical;

In[389]:=
ChangeComponents[Y[-{a, polar}, -{b, polar}], Y[{a, polar}, {b, polar}]]

   Added dependent rule Y^{1 1} → 0 for tensor Y

   Added independent rule Y^{1 2} → Y^{1 2} for tensor Y

   Added dependent rule Y^{2 1} → -Y^{1 2} for tensor Y

   Added dependent rule Y^{2 2} → 0 for tensor Y

   Added independent rule Y^1_1 → g_{1 2} Y^{1 2} for tensor Y

   Added dependent rule Y^2_1 → -Y_1^2 for tensor Y

   Added independent rule Y_1^2 → g_{1 1} Y^{1 2} for tensor Y

   Added independent rule Y^1_2 → g_{2 2} Y^{1 2} for tensor Y

   Added independent rule Y^2_2 → -g_{1 2} Y^{1 2} for tensor Y

   Added dependent rule Y_1^1 → -Y^1_1 for tensor Y

   Added dependent rule Y_2^1 → -Y^1_2 for tensor Y

   Added dependent rule Y_2^2 → -Y^2_2 for tensor Y

   Computed Y^a_b → g_{b c} Y^{a c} in 0.536889 Seconds

   Added dependent rule Y_{1 1} → 0 for tensor Y

   Added independent rule Y_{1 2} → g_{1 1} Y^1_2 + g_{1 2} Y^2_2 for tensor Y

   Added dependent rule Y_{2 1} → -Y_{1 2} for tensor Y

   Kept old independent rule Y_{1 2} → g_{1 1} Y^1_2 + g_{1 2} Y^2_2
     vs. new Y_{1 2} → g_{2 2} Y_1^2 - g_{1 2} Y^1_1 for tensor Y

   Added dependent rule Y_{2 2} → 0 for tensor Y

   Computed Y_{a b} → g_{a c} Y^c_b in 0.310710 Seconds

Out[389]=
FoldedRule[{Y_{1 1} → 0, Y_{2 1} → -Y_{1 2}, Y_{2 2} → 0},
 {Y_{1 2} → g_{1 1} Y^1_2 + g_{1 2} Y^2_2}, {Y^2_1 → -Y_1^2, Y_1^1 → -Y^1_1, Y_2^1 → -Y^1_2, Y_2^2 → -Y^2_2},
 {Y^1_1 → g_{1 2} Y^{1 2}, Y_1^2 → g_{1 1} Y^{1 2}, Y^1_2 → g_{2 2} Y^{1 2}, Y^2_2 → -g_{1 2} Y^{1 2}}]
```

The system prints `Value g_{1 2} Y^1_2 + g_{2 2} Y^2_2 expected to be 0 by symmetry`. This is because of `$UseValues = ToCanonical` which tells the system only to use the independent rules at each step. If we had written `$UseValues = All` instead that message would not have appeared. However, whenever a component is 0 because of the symmetry, the symmetry value is used, so our result is correct.

©2003–2004 José M. Martín–García

```
In[390]:=
TensorValues[Y]
```

```
Out[390]=
FoldedRule[{Y_{1 1} → 0, Y_{2 1} → -Y_{1 2}, Y_{2 2} → 0},
 {Y_{1 2} → g_{1 1} Y^1_2 + g_{1 2} Y^2_2}, {Y^2_1 → -Y_1^2, Y_1^1 → -Y^1_1, Y_2^1 → -Y^1_2, Y_2^2 → -Y^2_2},
 {Y^1_1 → g_{1 2} Y^{1 2}, Y_1^2 → g_{1 1} Y^{1 2}, Y^1_2 → g_{2 2} Y^{1 2}, Y^2_2 → -g_{1 2} Y^{1 2}},
 {Y^{1 1} → 0, Y^{2 1} → -Y^{1 2}, Y^{2 2} → 0}, {Y^{1 2} → Y^{1 2}}]
```

We can always collapse the FoldedRules, but this may generate very long expressions

```
In[391]:=
?CollapseFoldedRule
```

```
CollapseFoldedRule[frule, {n, m}] converts the elements n to m of the
  foldedrule frule into a single list of replacements in terms of the
  independent objects at level m. The positional argument follows the
  notation of Take: both n, m or just m can be negative, counting
  from the end; a single positive integer m represents {1, m}; a
  single negative integer -n represents {-n, -1}. A third argument
  can be given specifying a function to be mapped on all rules after
  collapsing a new level. CollapseFoldedRule[frule] or CollapseFoldedRule[
  frule, All] are interpreted as collapse of all elements of frule.
```

```
In[392]:=
CollapseFoldedRule[%%, All]
```

```
Out[392]=
FoldedRule[{Y_{1 1} → 0, Y_{2 1} → g_{1 2}^2 Y^{1 2} - g_{1 1} g_{2 2} Y^{1 2}, Y_{2 2} → 0, Y_{1 2} → -g_{1 2}^2 Y^{1 2} + g_{1 1} g_{2 2} Y^{1 2},
  Y^2_1 → -g_{1 1} Y^{1 2}, Y_1^1 → -g_{1 2} Y^{1 2}, Y_2^1 → -g_{2 2} Y^{1 2}, Y_2^2 → g_{1 2} Y^{1 2}, Y^1_1 → g_{1 2} Y^{1 2},
  Y_1^2 → g_{1 1} Y^{1 2}, Y^1_2 → g_{2 2} Y^{1 2}, Y^2_2 → -g_{1 2} Y^{1 2}, Y^{1 1} → 0, Y^{2 1} → -Y^{1 2}, Y^{2 2} → 0, Y^{1 2} → Y^{1 2}}]
```

```
In[393]:=
CNumbersOf[polar] ^= {0, 1, 2}
```

```
Out[393]=
{0, 1, 2}
```

We can also change bases

```
In[394]:=
ChangeComponents[v[{a, cartesian}], v[{a, polar}]]
```

```
Added independent rule v^0 → v^0 for tensor v

Added independent rule v^1 → v^1 for tensor v

Added independent rule v^2 → v^2 for tensor v

Added independent rule v^0 → e^0_0 v^0 + e^0_1 v^1 + e^0_2 v^2 for tensor v

Added independent rule v^1 → e_0^1 v^0 + e^1_1 v^1 + e^1_2 v^2 for tensor v

Added independent rule v^2 → e_0^2 v^0 + e_1^2 v^1 + e^2_2 v^2 for tensor v

Computed v^a → e^a_b v^b in 0.293718 Seconds
```

```
Out[394]=
FoldedRule[{},
 {v^0 → e^0_0 v^0 + e^0_1 v^1 + e^0_2 v^2, v^1 → e_0^1 v^0 + e^1_1 v^1 + e^1_2 v^2, v^2 → e_0^2 v^0 + e_1^2 v^1 + e^2_2 v^2}]
```

©2003–2004 José M. Martín–García

7.8. Replacing TensorValues in expressions

ToValues	Replace tensor values in a given expression

Replacing TensorValues in expressions

As we have seen, xCoba` does not provide an automatic tool for setting values (use Set or SetDelayed instead of rules), because this is potentially dangerous and removing the definitions would not be trivial. However, it is very simple to apply all the rules in TensorValues to a given expression

```
In[395]:=
?ToValues

    ToValues[expr] returns expr after replacing all known tensor-values for
      the tensors in the expression. ToValues[expr, tensor] replaces only
      values for the given tensor. ToValues[expr, list] replaces values
      for the tensors in the list. ToValues[expr, list, f] applies the
      function f after replacement of each of the tensors in the list.
```

```
In[396]:=
TensorValues[T, {{polar, polar}}]
```

```
Out[396]=
FoldedRule[{T^10 → T^01, T^20 → T^02, T^21 → T^12},
 {T^00 → 2, T^01 → 3, T^02 → 4, T^11 → 4, T^12 → 5, T^22 → 6}]
```

```
In[397]:=
$CVVerbose = False;
```

```
In[398]:=
ChangeComponents[T[-{a, polar}, -{b, polar}], T[{a, polar}, {b, polar}]]

    Computed T^a_b → g_bc T^ac in 0.052983 Seconds

    Computed T_ab → g_ac T^c_b in 0.034351 Seconds
```

```
Out[398]=
FoldedRule[{T_10 → T_01, T_20 → T_02, T_21 → T_12},
 {T_12 → 5, T_01 → 6, T_00 → g_01 T_0^1 + g_02 T_0^2 + g_00 T^0_0, T_02 → g_00 T^0_2 + g_01 T^1_2 + g_02 T^2_2,
  T_11 → g_01 T^0_1 + g_12 T_1^2 + g_11 T^1_1, T_22 → g_02 T^0_2 + g_12 T^1_2 + g_22 T^2_2},
 {T^1_0 → T_0^1, T^2_0 → T_0^2, T^2_1 → T_1^2, T^0_0 → T^0_0, T_1^0 → T^0_1, T_1^1 → T^1_1, T_2^0 → T^0_2, T_2^1 → T^1_2, T_2^2 → T^2_2},
 {T^0_0 → g_00 T^00 + g_01 T^01 + g_02 T^02, T_0^1 → g_00 T^01 + g_01 T^11 + g_02 T^12,
  T_0^2 → g_00 T^02 + g_01 T^12 + g_02 T^22, T^0_1 → g_01 T^00 + g_11 T^01 + g_12 T^02,
  T^1_1 → g_01 T^01 + g_11 T^11 + g_12 T^12, T_1^2 → g_01 T^02 + g_11 T^12 + g_12 T^22, T^0_2 → g_02 T^00 + g_12 T^01 + g_22
  T^1_2 → g_02 T^01 + g_12 T^11 + g_22 T^12, T^2_2 → g_02 T^02 + g_12 T^12 + g_22 T^22}]
```

```
In[399]:=
T[{a, polar}, {b, polar}] T[-{a, polar}, -{b, polar}]
```

```
Out[399]=
T_ab T^ab
```

```
In[400]:=
TraceBasisDummy[%]
```

```
Out[400]=
T_00 T^00 + T_01 T^01 + T_02 T^02 + T_10 T^10 + T_11 T^11 + T_12 T^12 + T_20 T^20 + T_21 T^21 + T_22 T^22
```

©2003–2004 José M. Martín–García

That expression has only the tensor T and so we get

```
In[401]:=
ToValues[%]

Out[401]=
86 + 2 (g00 (2 g00 + 3 g01 + 4 g02) + g01 (3 g00 + 4 g01 + 5 g02) + g02 (4 g00 + 5 g01 + 6 g02)) +
 4 (g01 (2 g01 + 3 g11 + 4 g12) + g11 (3 g01 + 4 g11 + 5 g12) + g12 (4 g01 + 5 g11 + 6 g12)) +
 8 (g00 (2 g02 + 3 g12 + 4 g22) + g01 (3 g02 + 4 g12 + 5 g22) + g02 (4 g02 + 5 g12 + 6 g22)) +
 6 (g02 (2 g02 + 3 g12 + 4 g22) + g12 (3 g02 + 4 g12 + 5 g22) + g22 (4 g02 + 5 g12 + 6 g22))
```

Now the metric tensors are explicit and hence:

```
In[402]:=
ToValues[%]

Out[402]=
574
```

■ 8. Charts

Thus far, everything is a field on a manifold or a function of a parameter. We shall want to express scalar fields as functions of coordinate fields. To do this we need a chart

DefChart	Define a basis
$Bases	List of currently defined bases
ChartQ	Check existence of a given basis name

Definition of a charts.

This whole section has not been implemented yet. At the moment, there is no easy way to restrict fields to points, for example. DefChart, however, does work. We need to supply a list of scalars (the coordinates)

```
In[403]:=
DefChart[cart, M3, {1, 2, 3}, {x[], y[], z[]}, BasisColor → Green]

   ** DefTensor: Defining tensor x[].

   ** DefTensor: Defining tensor y[].

   ** DefTensor: Defining tensor z[].

   ** DefCovD: Defining parallel derivative PDcart[-a].

   ** DefTensor: Defining vanishing torsion tensor TorsionPDcart[a, -b, -c].

   ** DefTensor: Defining
    symmetric Christoffel tensor ChristoffelPDcart[a, -b, -c].

   ** DefTensor: Defining vanishing Riemann tensor RiemannPDcart[-a, -b, -c, d].

   ** DefTensor: Defining vanishing Ricci tensor RicciPDcart[-a, -b].

   ** DefTensor: Defining antisymmetric +1 density etaUpcart[a, b, c].

   ** DefTensor: Defining antisymmetric -1 density etaDowncart[-a, -b, -c].
```

©2003–2004 José M. Martín–García

Notice that now the torsion tensor vanishes and the Christoffel is symmetric.

```
In[404]:=
     TorsionPDcart[a, -b, -c]

Out[404]=
     0

In[405]:=
     Bracket[a][Basis[s, {1, -cart}], Basis[s, {2, -cart}]]

Out[405]=
     0

In[406]:=
     ChartsOfManifold[M3]

Out[406]=
     {cart}

In[407]:=
     ChartQ /@ {cart, polar}

Out[407]=
     {True, False}

In[408]:=
     BasisQ /@ {cart, polar}

Out[408]=
     {True, True}

In[409]:=
     PD[-a][z[]]

Out[409]=
     e_a^3
```

■ 9. Computing tensor values

■ 10. Final comments

Note: For further information about `xCoba`', and to be kept informed about new releases, you may contact the authors electronically at yllanes@lattice.fis.ucm.es or jmm@iem.cfmac.csic.es. Suggestions and comments are welcome and very much appreciated!

This is xCobaDoc.nb, the docfile of `xCoba`', currently in version 0.6.3.

©2003–2004 José M. Martín–García

In[410]:=
```
? xAct`xCoba`*
```

xAct‘xCoba‘

AbsDet	CobaArray	epsilonToetaDown	ScalarsOfChart
AllComponentValues	ComponentArray	epsilonToetaUp	SeparateBasis
AutomaticBasisContractionStart	ComponentValue	etaDown	SetDaggerMatri
AutomaticBasisContractionStop	ContractBasis	etaDownToepsilon	SmallCircle
BasisArray	ContractFirst	etaUp	TableOfCompone
BasisColor	Coordinate	etaUpToepsilon	TensorValIDs
BasisOfCovD	DaggerCIndex	FormatBasis	TensorValues
BasisValues	DateOfValID	FreeToBasis	ThreadComponen
ChangeChart	DefBasis	HeldComponentValue	ToBasis
ChangeComponents	DefChart	InChart	ToValues
ChartColor	DeleteTensorValues	ManifoldOfChart	TraceBasisDumm
ChartsOfManifold	DependenciesOfBasis	MetricInBasis	UndefBasis
CNumbersOf	Disclaimer	PDOfBasis	UndefChart
Coba	DummyToBasis	RicciRotation	ValID

©2003–2004 José M. Martín–García