AntiSymmetricStandardOrder

■ `AntiSymmetricStandardOrder[x][expr]` allows to order vertical couples {flavor@i,Void}, ... of indices, irrespective of their flavor, in any antisymmetric tensor x present in *expr*. Upper indices are ordered first in alphabetic order on the left, then lower indices. It has some common behaviour with SymmetrizeSlots to which it is complementary (here with symmetry specification -1).

`AntiSymmetricStandardOrder[x, {ind}][expr]` orders the first 'ind' antisymmetrical indices.

`AntiSymmetricStandardOrder[x, {ind1, ind2}][expr]` orders the antisymmetrical indices between columns 'ind1' and 'ind2'.

■ See also: `SymmetrizeSlots`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]

In[6]:= DefineTensorShortcuts[{T, 2}, {T, 5}]

In[7]:= {Tddduu[red@k, h, blue@l, j, red@i], Tudduu[red@k, h, blue@l, j, red@i],
         Tuuuuu[red@k, h, blue@l, j, red@i], Tddddd[red@k, h, blue@l, j, red@i],
         Tuuuud[red@k, h, blue@l, j, red@i], Tudddd[red@k, h, blue@l, j, red@i]}

        % // AntiSymmetricStandardOrder[T]
        %% // AntiSymmetricStandardOrder[T, {2}]
        %%% // AntiSymmetricStandardOrder[T, {2, 4}]
```

$$\text{Out[7]}= \left\{T_{khl}{}^{ji}, T^{k}{}_{hl}{}^{ji}, T^{khlji}, T_{khlji}, T^{khlj}{}_{i}, T^{k}{}_{hlji}\right\}$$

$$\text{Out[8]}= \left\{T^{ij}{}_{hkl}, -T^{ijk}{}_{hl}, T^{hijkl}, T_{hijkl}, -T^{hjkl}{}_{i}, -T^{k}{}_{hijl}\right\}$$

$$\text{Out[9]}= \left\{-T_{hkl}{}^{ji}, T^{k}{}_{hl}{}^{ji}, -T^{hklji}, -T_{hkljli}, -T^{hklj}{}_{i}, T^{k}{}_{hlji}\right\}$$

$$\text{Out[10]}= \left\{T_{k}{}^{j}{}_{hl}{}^{i}, T^{kj}{}_{hl}{}^{i}, -T^{khjli}, -T_{khjli}, -T^{khjl}{}_{i}, -T^{k}{}_{hjli}\right\}$$

Antisymmetrization can be applied to the indices which are not derivation indicies

```
In[11]:= ContravariantD[CovariantD[Tdd[red@k, blue@h], red@l], {red@j, red@i}]
         % // AntiSymmetricStandardOrder[T, {2}]
```

$$\text{Out[11]}= T_{kh;l}{}^{;ji}$$

$$\text{Out[12]}= -T_{hk;l}{}^{;ji}$$

For derivation indicies, we must know what we are doing (here, antisymmetry in hk, and in ij)

```
In[13]:= ContravariantD[CovariantD[Tdd[red@k, blue@h], red@l], {red@j, red@i}]
         % // AntiSymmetricStandardOrder[T]
```

$$\text{Out[13]}= T_{kh;l}{}^{;ji}$$

$$\text{Out[14]}= T_{hk;l}{}^{;ij}$$

```
In[15]:= ClearTensorShortcuts[{T, 2}, {T, 5}]

In[16]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

BasisChange

■ `BasisChange[β, toflavor][expr]` allows to perform the change via the transformation β, of arbitrary combinations of tensors in an initial flavor, to the final flavor 'toflavor'.

■ See also: IndexFlavors, ToFlavor, Flavor.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]

In[6]:= DefineTensorShortcuts[{{T, e}, 1}, {T, 2}, {T, 3}]

In[7]:= Tu[red@i] // BasisChange[β, blue]

Out[7]= T^i β_i^i

In[8]:= Tuuu[red@i, red@j, red@k] // BasisChange[β, blue]

Out[8]= T^ijk β_i^i β_j^j β_k^k

In[9]:= f[Td[red@i]] // BasisChange[β, blue]

Out[9]= f(T_i β^i_i)

In[10]:= a Td[i] + b Tuud[red@i, red@j, red@k] // BasisChange[β, blue]

Out[10]= a T_i β^i_i + b T^ij_k β_i^i β_j^j β^k_k

In[11]:= Power[(a Td[i] + b Tuud[red@i, red@j, red@k]), 3] // BasisChange[β, blue]

Out[11]= (a T_i β^i_i + b T^ij_k β_i^i β_j^j β^k_k)^3

In[12]:= f[Tuud[red@i, red@j, red@k]] // BasisChange[β, blue]

Out[12]= f(T^ij_k β_i^i β_j^j β^k_k)

In[13]:= MapThread[{# // ToFlavor[red, blue], "⟸", # // (BasisChange[β, red])} &,
          {{Tensor[T, {blue@i, blue@j}, {Void, Void}], Tensor[T, {blue@i, Void}, {Void, blue@j}],
            Tensor[T, {Void, Void, Void}, {blue@i, blue@j, blue@k}],
            Tensor[T, {blue@i, blue@j, blue@k}, {Void, Void, Void}],
            Tensor[T, {blue@i, Void, blue@j}, {Void, blue@k, Void}]}}]
```

$$\text{Out[13]}= \begin{pmatrix} T^{ij} & \Longleftarrow & T^{ij}\,\beta_i^{\ i}\,\beta_j^{\ j} \\ T^{i}_{\ j} & \Longleftarrow & T^{i}_{\ j}\,\beta_i^{\ i}\,\beta^j_{\ j} \\ T_{ijk} & \Longleftarrow & T_{ijk}\,\beta^i_{\ i}\,\beta^j_{\ j}\,\beta^k_{\ k} \\ T^{ijk} & \Longleftarrow & T^{ijk}\,\beta_i^{\ i}\,\beta_j^{\ j}\,\beta_k^{\ k} \\ T^{i}_{\ k}{}^{j} & \Longleftarrow & T^{i}_{\ k}{}^{j}\,\beta_i^{\ i}\,\beta_j^{\ j}\,\beta^k_{\ k} \end{pmatrix}$$

Scalar products of basis vectors in different basis, and chains of basis changes :

```
In[14]:= ((eu[red@i].ed[j] // EvaluateDotProducts[e, β])
           (eu[j].ed[blue@k] // EvaluateDotProducts[e, β]))
         % // KroneckerAbsorb[β]

Out[14]= β^j_k β^i_j

Out[15]= β^i_k
```

Restore the settings.

```
In[16]:= ClearTensorShortcuts[{{T, e}, 1}, {T, 2}, {T, 3}]

In[17]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

BasisDerivation

■ BasisDerivation[Γ,m][{e,i},j][expr] expresses the partial derivative with respect to index j, of a basis vector Tensor[e, {Void}, {i}] present in expr, into a combination of basis vectors. The coefficients are Christoffel symbols Γ; m is a dummy index. The parameters are defined in labels = {x, δ, g, Γ}. This function includes the assignement to zero

of the covariant derivative of basis vectors.

- `BasisDerivation[Γ,h,m][{e,i},{j,k}][expr]` do the same kind of expansion for second partial derivatives. It includes the assignement to zero of the covariant derivatives of basis vectors."

- See also: `PartialD`, `ChristoffeluSymbol`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]

In[6]:= DefineTensorShortcuts[{e, 1}]

In[7]:= PartialD[ed[red@i], red@j]
        Print["                    is  equal  to "]
        (PartialD[ed[red@i], red@j] // BasisDerivation[Γ, red@m][{e, red@i}, red@j])
```

$$\text{Out[7]= } \mathfrak{e}_{i,j}$$

is equal to

$$\text{Out[9]= } \mathfrak{e}_m \, \Gamma^m{}_{ji}$$

and equivalently,

```
In[10]:= PartialD[eu[red@i], red@j]
         (PartialD[eu[red@i], red@j] // BasisDerivation[Γ, red@m][{e, red@i}, red@j])
```

$$\text{Out[10]= } \mathfrak{e}^i{}_{,j}$$

$$\text{Out[11]= } -\mathfrak{e}^m \, \Gamma^i{}_{jm}$$

In the case of second derivatives,

```
In[12]:= PartialD[ed[red@i], {red@j, red@k}] == (PartialD[ed[red@i], {red@j, red@k}] //
            BasisDerivation[Γ, red@h, red@m][{e, red@i}, {red@j, red@k}])
```

$$\text{Out[12]= } \mathfrak{e}_{i,jk} = \mathfrak{e}_h \left(\Gamma^h{}_{ji,k} + \Gamma^h{}_{km} \, \Gamma^m{}_{ji} \right)$$

```
In[13]:= PartialD[eu[red@i], {red@j, red@k}] == (PartialD[eu[red@i], {red@j, red@k}] //
            BasisDerivation[Γ, red@h, red@m][{e, red@i}, {red@j, red@k}])
```

$$\text{Out[13]= } \mathfrak{e}^i{}_{,jk} = \mathfrak{e}^h \left(\Gamma^i{}_{jm} \, \Gamma^m{}_{kh} - \Gamma^i{}_{jh,k} \right)$$

Restore the settings.

```
In[14]:= ClearTensorShortcuts[{e, 1}]

In[15]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

ChristoffeldSymbol

- `ChristoffeldSymbol[e,Γ,k][expr]` expresses the partial derivatives PartialD[Tensor[e, {Void}, {i}],j] of basis vectors 'e' on the basis using 'Γ'-noted Christoffel symbols with down indices; k is the dummy index of Einstein summation. We have chosen here to order the symmetrical Christoffel indices."

- See also: `PartialD`, `ChristoffeluSymbol`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]

In[6]:= DefineTensorShortcuts[{e, 1}]

In[7]:= PartialD[ed[red@i], red@j]
        % // ChristoffeldSymbol[e, Γ, k]

Out[7]= e_{i,j}

Out[8]= e^k Γ_{kij}
```

Restore the settings.

```
In[9]:= ClearTensorShortcuts[{e, 1}]

In[10]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

ChristoffeluSymbol

■ `ChristoffeluSymbol[e,Γ,k][expr]` expresses the partial derivatives PartialD[Tensor[e, {Void}, {i}],j] of basis vectors 'e' on the basis using 'Γ'-noted Christoffel symbols with up indices; k is the dummy index of Einstein summation. We have chosen here to order the symmetrical Christoffel indices."

■ See also: `PartialD`, `ChristoffeldSymbol`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]

In[6]:= DefineTensorShortcuts[{e, 1}]

In[7]:= PartialD[ed[red@i], red@j]
        % // ChristoffeluSymbol[e, Γ, k]

Out[7]= e_{i,j}

Out[8]= e_k Γ^k_{ij}

In[9]:= PartialD[eu[red@i], red@j]
        % // ChristoffeluSymbol[e, Γ, k]

Out[9]= e^i_{,j}

Out[10]= -e^k Γ^i_{jk}
```

Restore the settings.

```
In[11]:= ClearTensorShortcuts[{e, 1}]

In[12]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

ContravariantD2d

■ `ContravariantD2d[expr, i]` represents the 2-dimensional contravariant derivative of the tensor expression with respect to the index `i`.

■ `ContravariantD2d[expr, {i, j, ...}]` represents the 2-dimensional contravariant derivative of the expression with respect to the list of indices.

■ Contravariant two-dimensional derivatives are expanded as does Covariant derivatives. HoldContravariantD2d must be used to leave the expression unexpanded (Hold Form)

■ Each contravariant index is prefixed by a Alternatives (|) in the output display. The format can be changed using `SetDerivativeSymbols`.

■ The intended flavors must be on the indices in `ContravariantD2d`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[7]:= DefineTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]

In[8]:= Td[red@i]
        Print["The default general contravariant derivation is written,"]
        ContravariantD[Td[red@i], red@k]
        Print["          ...while the default 2d contravariant derivation is written :"]
        ContravariantD2d[Td[red@i], red@k]
```

Out[8]= T_i

The default general contravariant derivation is written,

Out[10]= $T_i^{;k}$

...while the default 2d contravariant derivation is written :

Out[12]= $T_i^{|k}$

Normal linear and Liebnizian differentiation rules are applied,

```
In[13]:= {Su[i] + Tu[i], Su[i] Tu[j]} // TableForm
         ContravariantD2d[#, k] & /@ % // TableForm
```

Out[13]//TableForm=

$S^i + T^i$

$S^i\, T^j$

Out[14]//TableForm=

$S^{i|k} + T^{i|k}$

$T^{j|k}\, S^i + S^{i|k}\, T^j$

Symbols and numeric quantities are not differentiated.

```
In[15]:= 2 π a Su[i] Tu[j]
         ContravariantD2d[%, k]
```

Out[15]= $2\,a\,\pi\,S^i\,T^j$

Out[16]= $2\,a\,\pi\left(T^{j|k}\,S^i + S^{i|k}\,T^j\right)$

The flavor must be on the covariant index when differentiating flavored expressions.

```
In[17]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         ContravariantD2d[%, red@k]
```

Out[17]= $\frac{2\,a\,\pi\,S^i\,T^j}{\nu + 1}$

Out[18]= $\frac{2\,a\,\pi\left(T^{j|k}\,S^i + S^{i|k}\,T^j\right)}{\nu + 1}$

Restore settings.

```
In[19]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]

In[20]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

ContravariantD

- `ContravariantD[`*expr*, *i*`]` represents the contravariant derivative of the tensor expression with respect to the index `i`.
- `ContravariantD[`*expr*, {*i*, *j*, ...}`]` represents the contravariant derivative of the expression with respect to the list of indices.

- Contravariant derivatives are expanded as does Covariant derivatives. HoldContravariantD must be used if we want to leave the expression unexpanded (Hold Form)
- Each contravariant index is prefixed by a semi-colon in the output display. The format can be changed using `SetDerivativeSymbols`.
- The intended flavors must be on the indices in `ContravariantD`.
- In presence of contravariant derivatives `SetScalarSingleCovariantD` must be set to `False`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]

In[7]:= DefineTensorShortcuts[{{R, S, T}, 1}, {{R, S, T}, 2}]
```

Here is a tensor T and its contravariant derivative with respect to index k. In the FullForm we see that the contravariant index is simply wrapped in the header `Cov`. In the display it is prefixed with a semi-colon.

```
In[8]:= Td[i]
        ContravariantD[%, k]
        % // FullForm

Out[8]= T_i

Out[9]= T_i^;k

Out[10]//FullForm=
HoldContravariantD[Tensor[T, List[Void], List[black[i]]], black[k]]
```

Normal linear and Liebnizian differentiation rules are applied in contravariant derivations. Hold forms are obtained using **`HoldContravariantD`** :

```
In[11]:= {Su[i] + Tu[i], Su[i] Tu[j]} // TableForm
         ContravariantD[#, k] & /@ % // TableForm

Out[11]//TableForm=
S^i + T^i
S^i T^j

Out[12]//TableForm=
S^i;k + T^i;k
T^j;k S^i + S^i;k T^j
```

The flavor must be on the contravariant index when differentiating.

```
In[13]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         ContravariantD[%, blue@k]
```

$$\text{Out[13]}= \frac{2\,a\,\pi\,S^{i}\,T^{j}}{\nu+1}$$

$$\text{Out[14]}= \frac{2\,a\,\pi\left(T^{j}{}^{;k}\,S^{i}+S^{i}{}^{;k}\,T^{j}\right)}{\nu+1}$$

Restore settings.

```
In[21]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

```
In[22]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

CovariantD2d

- `CovariantD2d[expr, i]` represents the 2-dimensional covariant derivative of the tensor expression with respect to the index `i`.
- `CovariantD2d[expr, {i, j, ...}]` represents the 2-dimensional covariant derivative of the expression with respect to the list of indices.

- Normal linear and Liebnizian differentiation rules are applied.
- Each covariant index is prefixed by a Alternatives (|) in the output display. The format can be changed using `SetDerivativeSymbols`.
- The intended flavors must be on the indices in `CovariantD2d`.
- Covariant derivatives of scalar tensors may have a partial differentiation for the first index, depending on **`SetScalarSingle-CovariantD`**
- See also: `CovariantD`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]
```

```
In[7]:= DefineTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

Here is a tensor T and its covariant derivative with respect to index k. In the FullForm we see that the covariant derivative is written in an HoldForm.

```
In[8]:= Td[red@i]
        CovariantD2d[%, red@k]
        % // FullForm
```

$$\text{Out[8]}= T_{i}$$

$$\text{Out[9]}= T_{i\,|\,k}$$

```
Out[10]//FullForm=
HoldCovariantD2d[Tensor[T, List[Void], List[red[i]]], red[k]]
```

Normal linear and Liebnizian differentiation rules are applied.

```
In[11]:= {Su[i] + Tu[i], Su[i] Tu[j]} // ToFlavor[red]
         CovariantD2d[#, red@k] & /@ %
```

$$\text{Out[11]}= \{S^{i}+T^{i},\ S^{i}\,T^{j}\}$$

$$\text{Out[12]}= \left\{S^{i}{}_{|\,k}+T^{i}{}_{|\,k},\ T^{j}{}_{|\,k}\,S^{i}+S^{i}{}_{|\,k}\,T^{j}\right\}$$

Symbols and numeric quantities are not differentiated.

```
In[13]:= 2 π a Su[i] Tu[j]
         CovariantD2d[%, k]
```

Out[13]= $2\,a\,\pi\,S^i\,T^j$

Out[14]= $2\,a\,\pi\left(T^j{}_{|k}\,S^i + S^i{}_{|k}\,T^j\right)$

The flavor must be on the covariant index when differentiating flavored expressions.

```
In[15]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         CovariantD2d[%, red@k]
```

Out[15]= $\frac{2\,a\,\pi\,S^i\,T^j}{\nu+1}$

Out[16]= $\frac{2\,a\,\pi\left(T^j{}_{|k}\,S^i + S^i{}_{|k}\,T^j\right)}{\nu+1}$

Covariant derivatives of scalar tensors may have a partial differentiation for the first index, depending on **SetScalarSingleCovariantD**.

```
In[17]:= SetScalarSingleCovariantD[False]
         Tensor[ϕ]
         CovariantD2d[%, red@i]
         CovariantD2d[%, red@j]
```

Out[18]= ϕ

Out[19]= $\phi_{|i}$

Out[20]= $\phi_{|i\,j}$

```
In[21]:= SetScalarSingleCovariantD[True]
         Tensor[ϕ]
         CovariantD2d[%, red@i]
         CovariantD2d[%, red@j]
```

Out[22]= ϕ

Out[23]= $\phi_{,i}$

Out[24]= $\phi_{|i\,j}$

Restore settings.

```
In[25]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

```
In[26]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

CovariantDOrdering

■ `CovariantDOrdering[expr]` orders the indices of covariant derivation in expr, in a standard order. This is only possible in spaces with zero Gaussian curvature, or for second order derivatives of scalar tensors..

■ A warning is printed to point out that the commutation of indicies in covariant derivatives is not always possible.

■ SymmetricStandardOrder can also be used to order (symmetrically) the indicies (there is no warning in this case)

■ See also: `SymmetricStandardOrder`, `PartialDOrdering`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[7]:= DefineTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

Case of scalar tensors

```
In[8]:= Tensor[T]
        CovariantD[%, {red@k, red@j, red@i}]
        % // CovariantDOrdering

Out[8]= T

Out[9]= T_;k j i

Out[10]= T_;i j k
```

Higher order tensors

```
In[11]:= Td[red@m]
         CovariantD[%, {red@k, red@j, red@i}]
         % // CovariantDOrdering

Out[11]= T_m

Out[12]= T_m ;k j i
```

We can use **CovariantDOrdering** if the space has a zero Gaussian curvature !

```
Out[13]= T_m ;i j k

In[14]:= Tuu[red@n, red@m]
         CovariantD[%, {red@k, red@j, red@i}]
         % // CovariantDOrdering

Out[14]= T^n m

Out[15]= T^n m_;k j i
```

We can use **CovariantDOrdering** if the space has a zero Gaussian curvature !

```
Out[16]= T^n m_;i j k
```

SymmetricStandardOrder can also be used to order (symmetrically) the indicies (there is no warning in this case) :

```
In[17]:= CovariantD[Tuu[red@n, red@m], {red@k, red@j, red@i}]
         % // SymmetricStandardOrder[T, {2}]
         % // SymmetricStandardOrder[T, {3, 5}]

Out[17]= T^n m_;k j i

Out[18]= T^m n_;k j i

Out[19]= T^m n_;i j k
```

Restore settings.

```
In[20]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]

In[21]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

CovariantDSimplify

- `CovariantDSimplify[e,g,e][expr]` assigns the value zero to Covariant or Contravariant derivatives of the basis vectors e, the metric tensor g and the Levi-Civita tensor e.
- `CovariantDSimplify[g][expr]` assigns the value zero to Covariant or Contravariant derivatives of the metric tensor g.
- For convenience, we have also defined `CovariantDSimplify[g,1][expr]` which commutes the covariant or contravariant derivatives with the metric tensor (putting the metric tensor inside the derivation, see the example).

■ See also: CovariantD, MetricSimplifyD

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]

In[6]:= DefineTensorShortcuts[{{u, v, e}, 1}, {{g, a}, 2}]
        SymbolSpaceDimension[a] = 2;

In[8]:= u = ud[red@i] eu[red@i]
        v = vu[red@j] ed[red@j]

Out[8]= u_i e^i

Out[9]= v^j e_j

In[10]:= CovariantD[u, red@k] vu[red@k]
         % // CovariantDSimplify[e, g, e]

Out[10]= v^k (e^i_;k u_i + u_i;k e^i)

Out[11]= u_i;k v^k e^i
```

CovariantDSimplify also simplifies the contravariant derivatives,

```
In[12]:= ContravariantD[u, red@k] vd[red@k]
         % // CovariantDSimplify[e, g, e]

Out[12]= v_k (e^i:k u_i + u_i^:k e^i)

Out[13]= u_i^:k v_k e^i
```

The operation can be applied after dummy indices expansion

```
In[14]:= CovariantD[u, red@k] vu[red@k] // EinsteinSum[];
         % // CovariantDSimplify[e, g, e]

Out[15]= u_1;1 v^1 e^1 + u_1;2 v^2 e^1 + u_1;3 v^3 e^1 + u_2;1 v^1 e^2 + u_2;2 v^2 e^2 + u_2;3 v^3 e^2 + u_3;1 v^1 e^3 + u_3;2 v^2 e^3 + u_3;3 v^3 e^3
```

To raise or lower the indices inside the covariant derivative of an expression, we can first use **CovariantDSimplify[g]** to commute the derivation and the metric tensor, and

then **MetricSimplifyD[g]** allows raising or lowering the indices inside the covariant derivative

```
In[16]:= HoldCovariantD[uu[red@i] vd[red@j], red@k] guu[red@j, red@h]
         % // CovariantDSimplify[g, 1]
         % // MetricSimplifyD[g]

Out[16]= (u^i v_j)_;k g^jh

Out[17]= (g^jh u^i v_j)_;k

Out[18]= (u^i v^h)_;k

In[19]:= HoldContravariantD[uu[red@i] vd[red@j], red@k] guu[red@j, red@h]
         % // CovariantDSimplify[g, 1]
         % // MetricSimplifyD[g]

Out[19]= (u^i v_j)^:k g^jh

Out[20]= (g^jh u^i v_j)^:k

Out[21]= (u^i v^h)^:k
```

In two-dimensions, if **a** is the associated metric tensor (It is then necessary to assign **SymbolSpaceDimension[a] = 2,** see above In[8]):

```
In[22]:= HoldCovariantD2d[uu[red@i] vd[red@j], red@k] auu[red@j, red@h]
         % // CovariantDSimplify[a, 1]
         % // MetricSimplifyD[a]
```

Out[22]= $(u^i v_j)_{|k}\, a^{jh}$

Out[23]= $\left(a^{jh} u^i v_j\right)_{|k}$

Out[24]= $(u^i v^h)_{|k}$

```
In[25]:= HoldContravariantD2d[uu[red@i] vd[red@j], red@k] aud[red@j, red@h]
         % // CovariantDSimplify[a, 1]
         % // MetricSimplifyD[a]
```

Out[25]= $(u^i v_j)^{|k}\, a^j{}_h$

Out[26]= $\left(a^j{}_h u^i v_j\right)^{|k}$

Out[27]= $(u^i v_h)^{|k}$

```
In[28]:= ClearTensorShortcuts[{{u, v, e}, 1}]
```

```
In[29]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

CrossProductExpansion

■ `CrossProductExpansion[BasisVectors,e][expr]` expands three dimensional CrossProducts and Triple Scalar Products of basis vectors appearing in expr, into Levi-Civita tensors noted 'e'.

■ See also: `LeviCivitaSimplify`, `LeviCivitaOrder`, `EvaluateCrossProducts`

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{blue, Blue}]
```

```
In[6]:= DefineTensorShortcuts[{{e}, 1}]
```

```
In[7]:= eu[blue@i] × eu[blue@j]
        % // CrossProductExpansion[e, e, k]
```

Out[7]= $\mathbf{e}^i \times \mathbf{e}^j$

Out[8]= $e^{ijk}\, \mathbf{e}_k$

```
In[9]:= (% // EinsteinSum[] // ArrayExpansion[blue@i, blue@j]) /. PermutationSymbolRule[e]
```

Out[9]= $\begin{pmatrix} 0 & \mathbf{e}_3 & -\mathbf{e}_2 \\ -\mathbf{e}_3 & 0 & \mathbf{e}_1 \\ \mathbf{e}_2 & -\mathbf{e}_1 & 0 \end{pmatrix}$

```
In[10]:= ClearTensorShortcuts[{{e}, 1}]
```

```
In[11]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

DifToCov

■ `DifToCov[Γ,m][{u,i},j][expr]` expresses the partial derivative with respect to index j, of a tensor u present in expr, into a combination of covariant derivative and components of u. The coefficients contain Christoffel symbols Γ with

m as a dummy index.

■ `DifToCov[Γ,m,h][{u,i},{j,k}][expr]` do the same kind of expansion for second partial derivatives.

■ See also: `PartialD`, `CovariantD`, `ChristoffeluSymbol`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]

In[6]:= DefineTensorShortcuts[{u, 1}]

In[7]:= PartialD[ud[red@i], red@j] ==
         (PartialD[ud[red@i], red@j] // DifToCov[Γ, red@m][{u, red@i}, red@j])
        PartialD[uu[red@i], red@j] ==
         (PartialD[uu[red@i], red@j] // DifToCov[Γ, red@m][{u, red@i}, red@j])
```

$$\text{Out[7]=}\quad u_{i}{}_{,j} = u_{i}{}_{;j} + u_{m}\,\Gamma^{m}{}_{i\,j}$$

$$\text{Out[8]=}\quad u^{i}{}_{,j} = u^{i}{}_{;j} - u^{m}\,\Gamma^{i}{}_{j\,m}$$

```
In[9]:= PartialD[uu[red@i], {red@j, red@k}] == (PartialD[uu[red@i], {red@j, red@k}] //
           DifToCov[Γ, red@p, red@q][{u, red@i}, {red@j, red@k}])
        PartialD[ud[red@i], {red@j, red@k}] == (PartialD[ud[red@i], {red@j, red@k}] //
           DifToCov[Γ, red@p, red@q][{u, red@i}, {red@j, red@k}])
```

$$\text{Out[9]=}\quad u^{i}{}_{,j\,k} = u^{i}{}_{;j\,k} - \Gamma^{i}{}_{j\,p}{}_{,k}\,u^{p} - u^{p}{}_{;k}\,\Gamma^{i}{}_{j\,p} - u^{q}{}_{;j}\,\Gamma^{i}{}_{q\,k} + u^{q}\,\Gamma^{i}{}_{j\,p}\,\Gamma^{p}{}_{q\,k} + u^{i}{}_{;q}\,\Gamma^{q}{}_{j\,k}$$

$$\text{Out[10]=}\quad u_{i}{}_{,j\,k} = u_{i}{}_{;j\,k} + \Gamma^{p}{}_{i\,j}{}_{,k}\,u_{p} + u_{p}{}_{;k}\,\Gamma^{p}{}_{i\,j} + u_{q}{}_{;j}\,\Gamma^{q}{}_{i\,k} + u_{i}{}_{;q}\,\Gamma^{q}{}_{j\,k} + u_{q}\,\Gamma^{p}{}_{i\,j}\,\Gamma^{q}{}_{k\,p}$$

```
In[11]:= ClearTensorShortcuts[{u, 1}]

In[12]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

DifToPartialD

■ `DifToPartialD[labels][expr]` rewrites the partial derivatives with respect to coordinates indices present into the partial derivatives in their expanded form parametrized by labels, it is equivalent to using `ExpandPartialD[labels][expr]`, but we defined this operation by symmetry with `PartialDToDif[expr]`."

■ See also: `PartialDToDif`, `PartialD`, `ExpandPartialD`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[6]:= DefineTensorShortcuts[{T, 2}, {T, 4}]

In[7]:= labels = {x, δ, g, Γ};
        (PartialD[Tuu[i, j], {k, h}]) == (PartialD[Tuu[i, j], {k, h}] // DifToPartialD[labels])
```

$$\text{Out[8]=}\quad T^{i\,j}{}_{,k\,h} = \frac{\partial^{2}\,T^{i\,j}}{\partial x^{k}\,\partial x^{h}}$$

`DifToPartialD[labels]` is equivalent to **`ExpandPartialD[labels]`**, we have conserved it by symmetry with its reverse **`PartialDToDif :`**

```
In[9]:= PartialD[Tuu[i, j], {k, h}] // ExpandPartialD[labels]
```

$$\text{Out[9]}= \frac{\partial^2 T^{i\,j}}{\partial x^k \partial x^h}$$

Restore the settings.

```
In[10]:= ClearTensorShortcuts[{T, 2}, {T, 3}]

In[11]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

EvaluateCrossProducts

■ `EvaluateCrossProducts[e,e,g,h][expr]` evaluates the cross products in expr. e designates the basis used, e is the Levi-Civita symbol, g the metric tensor, and h a dummy index (not present in expr).

■ **EvaluateCrossProducts** applies successively LinearBreakout[Cross][ed[_],eu[_]], CrossProductExpansion[e,e,h], FullLeviCivitaExpand[e,g], MetricSimplify[g], and finally replaces the trace of the metric tensor g by the space dimension NDim.

■ See also: `LeviCivitaSimplify`, `LeviCivitaOrder`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[6]:= DefineTensorShortcuts[{{e, t}, 1}]

In[7]:= t := tu[h] ed[h] // ToFlavor[red]

In[8]:= t × TCurl[e, g, red /@ {i, j, l}, e][t]
        % // EvaluateCrossProducts[e, e, g, red@k]
        %[[1]] + (%[[2]] /. {i → j, j → i}) // Simplify
```

$$\text{Out[8]}= (t^h \mathfrak{e}_h) \times \left(t_{j\,;i}\, e^{i\,j\,l}\, \mathfrak{e}_l\right)$$

$$\text{Out[9]}= t_{j\,;i}\, t^j\, \mathfrak{e}^i - t_{j\,;i}\, t^i\, \mathfrak{e}^j$$

$$\text{Out[10]}= \left(t_{j\,;i} - t_{i\,;j}\right) t^j\, \mathfrak{e}^i$$

```
In[11]:= TCurl[e, g, red /@ {i, j, k}, e][t] × TCurl[e, g, red /@ {p, q, r}, e][t]
         % // EvaluateCrossProducts[e, e, g, l] // LeviCivitaOrder[e] // Simplify
```

$$\text{Out[11]}= \left(t_{j\,;i}\, e^{i\,j\,k}\, \mathfrak{e}_k\right) \times \left(t_{q\,;p}\, e^{p\,q\,r}\, \mathfrak{e}_r\right)$$

$$\text{Out[12]}= t_{j\,;i}\, t_{q\,;p}\, (e^{i\,p\,q}\, \mathfrak{e}^j - e^{j\,p\,q}\, \mathfrak{e}^i)$$

```
In[13]:= ClearTensorShortcuts[{{e, t}, 1}]

In[14]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

ExpandCovariantD2d

■ `ExpandCovariantD2d[{x, δ, a, Γ}, i, permissive : False][expr]` will expand first order two-dimensional covariant derivatives of tensors using x as the label for the coordinate positions, δ as the label for the Kronecker, a as the label for the 2d metric tensor and Γ as the label for Christoffel symbols. The introduced dummy index will be i.

■ `ExpandCovariantD2d[{x, δ, a, Γ}, {i, j, ...}, permissive : False][expr]` expands higher order two-dimensional covariant derivatives using the list of dummy indices.

- ExpandCovariantD2d is mapped over arrays, equations and Plus.
- ExpandCovariantD2d is intended to work only on tensor expressions. It will not expand expressions containing base indices or unexpanded partial derivatives. It will not expand expressions containing expanded partial derivatives or Christoffel symbols (from Γ in the list of labels) unless the user sets the optional parameter permissive to True. This might be done if the expression contain results from previous covariant derivatives.
- The expansion is done for a 'coordinate basis' or holonomic system. in which case the Christoffel up symbols, with the first index up, are the same as the 'connection coefficients'.
- When working in a notebook with a constant set of labels one can put `labs = {x, δ, a, Γ}` and then use `ExpandCovariantD2d[labs, i][expr]` in the call, with similar usage for the other derivative routines.
- See also: `CovariantD2d`, `SetChristoffelValueRules`, `PartialD`, `AbsoluteD`, `TotalD`, `Tensor`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the old settings.

```
In[3]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]
```

Define standard labels and shortcuts.

```
In[7]:= Base2d = {1, 2};
        Base3d = {1, 2, 3};

        labsz = {xb, δ, g, Γb};
        labs0 = {x, δ, a, Γ};
        DefineTensorShortcuts[{{x, y, xb, S, T}, 1}, {{a, g, δ, S, T}, 2}, {{Γ, Γb, Λ, zero}, 3}]
        TensorLabelFormat[xb, OverBar[x]]
        TensorLabelFormat[Γb, OverBar[Γ]]
        DeclareZeroTensor[zero]
        TensorLabelFormat[zero, 0]
```

Here is a covariant derivative of a second order tensor and its expansion in terms of a partial derivatives and Christoffel symbols using the dummy index a. There is a Christoffel correction term for each index.

```
In[16]:= Tuu[i, j]
         CovariantD[%, j]
         % // ExpandCovariantD[labsz, g]
         % // EinsteinSum[]
```

```
Out[16]= T^{i j}
```

```
Out[17]= T^{i j}_{;j}
```

```
Out[18]= T^{g j} Γ^{i}_{j g} + T^{i g} Γ^{j}_{j g} + ∂T^{i j}/∂x^{j}
```

```
Out[19]= T^{i 1} Γ^{1}_{1 1} + T^{i 2} Γ^{1}_{1 2} + T^{i 3} Γ^{1}_{1 3} + T^{i 1} Γ^{2}_{2 1} + T^{i 2} Γ^{2}_{2 2} + T^{i 3} Γ^{2}_{2 3} +
          T^{i 1} Γ^{3}_{3 1} + T^{i 2} Γ^{3}_{3 2} + T^{i 3} Γ^{3}_{3 3} + T^{1 1} Γ^{i}_{1 1} + T^{2 1} Γ^{i}_{1 2} + T^{3 1} Γ^{i}_{1 3} + T^{1 2} Γ^{i}_{2 1} +
          T^{2 2} Γ^{i}_{2 2} + T^{3 2} Γ^{i}_{2 3} + T^{1 3} Γ^{i}_{3 1} + T^{2 3} Γ^{i}_{3 2} + T^{3 3} Γ^{i}_{3 3} + ∂T^{i 1}/∂x^{1} + ∂T^{i 2}/∂x^{2} + ∂T^{i 3}/∂x^{3}
```

```
In[20]:= SymbolSpaceDimension[Γ] = 2;
         SymbolSpaceDimension[a] = 2;
         SymbolSpaceDimension[Γb] = 3;
         SymbolSpaceDimension[g] = 3;
```

```
In[24]:= DeclareBaseIndices[Base2d]
         Tuu[i, j]
         CovariantD2d[%, j]
         % // ExpandCovariantD2d[labs0, h]
         % // EinsteinSum[]
```

$$\text{Out[25]=}\quad T^{i\,j}$$

$$\text{Out[26]=}\quad T^{i\,j}{}_{|\,j}$$

$$\text{Out[27]=}\quad T^{h\,j}\,\Gamma^{i}{}_{j\,h} + T^{i\,h}\,\Gamma^{j}{}_{j\,h} + \frac{\partial T^{i\,j}}{\partial x^{j}}$$

$$\text{Out[28]=}\quad T^{i\,1}\,\Gamma^{1}{}_{1\,1} + T^{i\,2}\,\Gamma^{1}{}_{1\,2} + T^{i\,1}\,\Gamma^{2}{}_{2\,1} + T^{i\,2}\,\Gamma^{2}{}_{2\,2} + T^{1\,1}\,\Gamma^{i}{}_{1\,1} + T^{2\,1}\,\Gamma^{i}{}_{1\,2} + T^{1\,2}\,\Gamma^{i}{}_{2\,1} + T^{2\,2}\,\Gamma^{i}{}_{2\,2} + \frac{\partial T^{i\,1}}{\partial x^{1}} + \frac{\partial T^{i\,2}}{\partial x^{2}}$$

while we cannot expand the 2-dimensional derivative using the 3-dimensional parameters :

```
In[29]:= Tuu[i, j]
         CovariantD2d[%, j]
         % // ExpandCovariantD2d[labsz, h]
```

$$\text{Out[29]=}\quad T^{i\,j}$$

$$\text{Out[30]=}\quad T^{i\,j}{}_{|\,j}$$

$$\text{Out[31]=}\quad T^{i\,j}{}_{|\,j}$$

With flavored expressions the intended flavor must also be on the covariant dummy index.

```
In[32]:= Tud[i, j] // ToFlavor[red]
         CovariantD2d[%, red@k]
         % // ExpandCovariantD2d[labs0, red@a]
```

$$\text{Out[32]=}\quad T^{i}{}_{j}$$

$$\text{Out[33]=}\quad T^{i}{}_{j\,|\,k}$$

$$\text{Out[34]=}\quad -T^{i}{}_{a}\,\Gamma^{a}{}_{k\,j} + T^{a}{}_{j}\,\Gamma^{i}{}_{k\,a} + \frac{\partial T^{i}{}_{j}}{\partial x^{k}}$$

A dummy index name must be supplied for each covariant differentiation.

```
In[35]:= Tuu[i, j] // ToFlavor[red]
         CovariantD2d[%, red /@ {m, n}]
         % // ExpandCovariantD2d[labs0, red /@ {a, b}]
```

$$\text{Out[35]=}\quad T^{i\,j}$$

$$\text{Out[36]=}\quad T^{i\,j}{}_{|\,m\,n}$$

$$\text{Out[37]=}\quad \Gamma^{i}{}_{m\,a}\,\frac{\partial T^{a\,j}}{\partial x^{n}} + \Gamma^{i}{}_{n\,b}\left(T^{a\,j}\,\Gamma^{b}{}_{m\,a} + T^{b\,a}\,\Gamma^{j}{}_{m\,a} + \frac{\partial T^{b\,j}}{\partial x^{m}}\right) + \Gamma^{j}{}_{m\,a}\,\frac{\partial T^{i\,a}}{\partial x^{n}} + \Gamma^{j}{}_{n\,b}\left(T^{i\,a}\,\Gamma^{b}{}_{m\,a} + T^{a\,b}\,\Gamma^{i}{}_{m\,a} + \frac{\partial T^{i\,b}}{\partial x^{m}}\right) +$$
$$\frac{\partial^{2} T^{i\,j}}{\partial x^{n}\,\partial x^{m}} - \Gamma^{b}{}_{n\,m}\left(T^{a\,j}\,\Gamma^{i}{}_{b\,a} + T^{i\,a}\,\Gamma^{j}{}_{b\,a} + \frac{\partial T^{i\,j}}{\partial x^{b}}\right) + T^{a\,j}\,\frac{\partial \Gamma^{i}{}_{m\,a}}{\partial x^{n}} + T^{i\,a}\,\frac{\partial \Gamma^{j}{}_{m\,a}}{\partial x^{n}}$$

We could use different symbols for the metric tensor and Christoffel symbols, but we have then to confirm that the new symbol is associated with the chosen space dimension

This uses Λ for the connection and y for the coordinates.

```
In[38]:= Tdd[i, j] // ToFlavor[red]
         CovariantD2d[%, red@k]
         % // ExpandCovariantD2d[{y, δ, G, Λ}, red@a]
         SymbolSpaceDimension[Λ] = 2; SymbolSpaceDimension[G] = 2;
         %% // ExpandCovariantD2d[{y, δ, G, Λ}, red@a]
```

$$\text{Out[38]=}\quad T_{i\,j}$$

$$\text{Out[39]=}\quad T_{i\,j\,|\,k}$$

$$\text{Out[40]=}\quad T_{i\,j\,|\,k}$$

$$\text{Out[42]=}\quad -T_{a\,j}\,\Lambda^{a}{}_{k\,i} - T_{i\,a}\,\Lambda^{a}{}_{k\,j} + \frac{\partial T_{i\,j}}{\partial y^{k}}$$

If a tensor has no covariant derivative indices then `ExpandCovariantD2d` does nothing. In the following, only one term is expanded.

```
In[43]:= 2 Tud[i, j] + CovariantD2d[Tu[i], j]
         % // ExpandCovariantD2d[labs0, a]
         % // ArrayExpansion[black /@ {i, j, a}] // MatrixForm
```

Out[43]= $T^{i}{}_{|j} + 2\,T^{i}{}_{j}$

Out[44]= $2\,T^{i}{}_{j} + T^{a}\,\Gamma^{i}{}_{ja} + \frac{\partial T^{i}}{\partial x^{j}}$

Out[45]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 2\,T^{1}{}_{1} + T^{1}\,\Gamma^{1}{}_{11} + \frac{\partial T^{1}}{\partial x^{1}} \\ 2\,T^{1}{}_{1} + T^{2}\,\Gamma^{1}{}_{12} + \frac{\partial T^{1}}{\partial x^{1}} \end{pmatrix} & \begin{pmatrix} 2\,T^{1}{}_{2} + T^{1}\,\Gamma^{1}{}_{21} + \frac{\partial T^{1}}{\partial x^{2}} \\ 2\,T^{1}{}_{2} + T^{2}\,\Gamma^{1}{}_{22} + \frac{\partial T^{1}}{\partial x^{2}} \end{pmatrix} \\ \begin{pmatrix} 2\,T^{2}{}_{1} + T^{1}\,\Gamma^{2}{}_{11} + \frac{\partial T^{2}}{\partial x^{1}} \\ 2\,T^{2}{}_{1} + T^{2}\,\Gamma^{2}{}_{12} + \frac{\partial T^{2}}{\partial x^{1}} \end{pmatrix} & \begin{pmatrix} 2\,T^{2}{}_{2} + T^{1}\,\Gamma^{2}{}_{21} + \frac{\partial T^{2}}{\partial x^{2}} \\ 2\,T^{2}{}_{2} + T^{2}\,\Gamma^{2}{}_{22} + \frac{\partial T^{2}}{\partial x^{2}} \end{pmatrix} \end{pmatrix}$$

The following is an example of a calculation with covariant derivatives. We set the metric for the surface of a 3d sphere in a spherical coordinate system and express it in terms of coordinate positions (x^3 is along the normal to the sphere).

```
In[46]:= metric = DiagonalMatrix[{R^2, R^2 Sin[θ]^2}];
         (tmetric = metric // CoordinatesToTensors[{θ, ϕ}]) // MatrixForm
         MapThread[SetTensorValueRules[#1, #2] &,
           {{add[i, j], auu[i, j]}, {tmetric, Inverse@tmetric}}];
```

Out[47]//MatrixForm=

$$\begin{pmatrix} R^2 & 0 \\ 0 & R^2\,\text{Sin}[x^1]^2 \end{pmatrix}$$

We then calculate and set the Christoffel values...

```
In[49]:= MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]}, CalculateChristoffels[labs0]}];
```

The following displays the independent nonzero values of the up Christoffel symbols.

```
In[50]:= PartialD[labs0][R, Tensor[x, {i_}, {Void}]] = 0;
         SelectedTensorRules[Γ, Γudd[_, a_, b_] /; OrderedQ[{a, b}]] // TableForm
```

Out[51]//TableForm=

$\Gamma^{1}{}_{22} \to -\text{Cos}[x^1]\,\text{Sin}[x^1]$

$\Gamma^{2}{}_{12} \to \text{Cot}[x^1]$

We then

1) Set the coordinate position values to coordinate symbols via rules.

2) Take the covariant derivative.

3) Expand the covariant derivative in terms of Christoffel symbols and the coordinate positions.

4) Expand the expression into an array making all substitutions.

5) Clear the rules for the coordinate positions.

```
In[52]:= SetTensorValueRules[xu[i], {θ, ϕ}];
         CovariantD2d[Tu[i], k]
         % // ExpandCovariantD2d[labs0, a]
         % // ToArrayValues[]
         ClearTensorValues[xu[i]]
```

Out[53]= $T^{i}{}_{|k}$

Out[54]= $T^{a}\,\Gamma^{i}{}_{ka} + \frac{\partial T^{i}}{\partial x^{k}}$

Out[55]= $\left\{\left\{\frac{\partial T^{1}}{\partial \theta}, -\text{Cos}[\theta]\,\text{Sin}[\theta]\,T^{2} + \frac{\partial T^{1}}{\partial \phi}\right\}, \left\{\text{Cot}[\theta]\,T^{2} + \frac{\partial T^{2}}{\partial \theta}, \text{Cot}[\theta]\,T^{1} + \frac{\partial T^{2}}{\partial \phi}\right\}\right\}$

That gives us the covariant derivative components of T in terms of the contravariant T components.

A tensor times a covariant derivative.

```
In[57]:= Tdu[k, j] CovariantD2d[Sd[k], m]
         % // ExpandCovariantD2d[labs0, a]
```

$$\text{Out[57]= } S_{k\,|\,m}\ T_k{}^j$$

$$\text{Out[58]= } T_k{}^j \left(-S_a\ \Gamma^a{}_{mk} + \frac{\partial S_k}{\partial x^m}\right)$$

The covariant derivative of a product of tensors and a constant.

```
In[59]:= 2 π q Sd[k] Tdu[j, k]
         CovariantD2d[%, m]
         % // ExpandCovariantD2d[labs0, a]
```

$$\text{Out[59]= } 2\,\pi\,q\ S_k\ T_j{}^k$$

$$\text{Out[60]= } 2\,\pi\,q\ \left(T_j{}^k{}_{|\,m}\ S_k + S_{k\,|\,m}\ T_j{}^k\right)$$

$$\text{Out[61]= } 2\,\pi\,q\ T_j{}^k \left(-S_a\ \Gamma^a{}_{mk} + \frac{\partial S_k}{\partial x^m}\right) + 2\,\pi\,q\ S_k \left(-T_a{}^k\ \Gamma^a{}_{mj} + T_j{}^a\ \Gamma^k{}_{ma} + \frac{\partial T_j{}^k}{\partial x^m}\right)$$

```
In[62]:= Sd[a] Tu[b]
         CovariantD2d[%, {c, d}]
         % // ExpandCovariantD2d[labs0, {e, f}]
```

$$\text{Out[62]= } S_a\ T^b$$

$$\text{Out[63]= } S_{a\,|\,d}\ T^b{}_{|\,c} + S_{a\,|\,c}\ T^b{}_{|\,d} + T^b{}_{|\,cd}\ S_a + S_{a\,|\,cd}\ T^b$$

$$\begin{aligned}\text{Out[64]= } & \left(-S_e\ \Gamma^e{}_{da} + \frac{\partial S_a}{\partial x^d}\right) \left(T^f\ \Gamma^b{}_{cf} + \frac{\partial T^b}{\partial x^c}\right) + \left(-S_e\ \Gamma^e{}_{ca} + \frac{\partial S_a}{\partial x^c}\right) \left(T^f\ \Gamma^b{}_{df} + \frac{\partial T^b}{\partial x^d}\right) + \\ & S_a \left(\frac{\partial^2\ T^b}{\partial x^d\ \partial x^c} - \Gamma^f{}_{dc} \left(T^e\ \Gamma^b{}_{fe} + \frac{\partial T^b}{\partial x^f}\right) + \Gamma^b{}_{ce}\ \frac{\partial T^e}{\partial x^d} + \Gamma^b{}_{df} \left(T^e\ \Gamma^f{}_{ce} + \frac{\partial T^f}{\partial x^c}\right) + T^e\ \frac{\partial\ \Gamma^b{}_{ce}}{\partial x^d}\right) + \\ & T^b \left(\frac{\partial^2\ S_a}{\partial x^d\ \partial x^c} - \Gamma^f{}_{dc} \left(-S_e\ \Gamma^e{}_{fa} + \frac{\partial S_a}{\partial x^f}\right) - \Gamma^e{}_{ca}\ \frac{\partial S_e}{\partial x^d} - \Gamma^f{}_{da} \left(-S_e\ \Gamma^e{}_{cf} + \frac{\partial S_f}{\partial x^c}\right) - S_e\ \frac{\partial\ \Gamma^e{}_{ca}}{\partial x^d}\right)\end{aligned}$$

Now, let's try the well known identity about the covariant derivative of the metric tensor, using the metric tensor and Christoffel symbols from above. We equate the covariant derivative to the 3rd order zero tensor.

```
In[65]:= SetTensorValueRules[δud[i, j], IdentityMatrix[NDim]]
         add[m, n]
         CovariantD2d[%, i] == zeroddd[n, m, i]
         step1 = % // ExpandCovariantD2d[labs0, a]
         ToArrayValues[] /@ % // MatrixForm
```

$$\text{Out[66]= } a_{mn}$$

$$\text{Out[67]= } a_{mn\,|\,i} == 0_{nmi}$$

$$\text{Out[68]= } -a_{an}\ \Gamma^a{}_{im} - a_{ma}\ \Gamma^a{}_{in} + \frac{\partial\ a_{mn}}{\partial x^i} == 0_{nmi}$$

Out[69]//MatrixForm=

True

The double covariant derivative of a scalar still requires two expansion indices, but the first one is not used and can even be a Null.

```
In[70]:= Tensor[ϕ]
         CovariantD2d[%, {a, b}]
         % // ExpandCovariantD2d[labs0, {, d}]
```

$$\text{Out[70]= } \phi$$

$$\text{Out[71]= } \phi_{|\,ab}$$

$$\text{Out[72]= } \frac{\partial^2\ \phi}{\partial x^b\ \partial x^a} - \Gamma^d{}_{ba}\ \frac{\partial \phi}{\partial x^d}$$

That usage allows us to use the same ExpandCovariantD2d on scalars and tensor expressions that evaluate to scalars.

```
In[73]:= Tensor[ϕ] + Su[a] Sd[a]
         CovariantD2d[%, {b, c}]
         % // ExpandCovariantD2d[labs0, {d, e}]
         % // TensorSimplify
         % // MapLevelParts[UpDownSwap[a], {{5, 6, 8}}]
```

$$\text{Out[73]=}\quad \phi + S_a\, S^a$$

$$\text{Out[74]=}\quad \phi_{|\,bc} + S_{a\,|\,c}\, S^a{}_{|\,b} + S_{a\,|\,b}\, S^a{}_{|\,c} + S^a{}_{|\,bc}\, S_a + S_{a\,|\,bc}\, S^a$$

$$\text{Out[75]=}\quad \frac{\partial^2 \phi}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb}\, \frac{\partial \phi}{\partial x^e} + \left(-S_d\, \Gamma^d{}_{ca} + \frac{\partial S_a}{\partial x^c}\right)\left(S^e\, \Gamma^a{}_{be} + \frac{\partial S^a}{\partial x^b}\right) + \left(-S_d\, \Gamma^d{}_{ba} + \frac{\partial S_a}{\partial x^b}\right)\left(S^e\, \Gamma^a{}_{ce} + \frac{\partial S^a}{\partial x^c}\right) +$$
$$S_a\left(\frac{\partial^2 S^a}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb}\left(S^d\, \Gamma^a{}_{ed} + \frac{\partial S^a}{\partial x^e}\right) + \Gamma^a{}_{bd}\, \frac{\partial S^d}{\partial x^c} + \Gamma^a{}_{ce}\left(S^d\, \Gamma^e{}_{bd} + \frac{\partial S^e}{\partial x^b}\right) + S^d\, \frac{\partial \Gamma^a{}_{bd}}{\partial x^c}\right) +$$
$$S^a\left(\frac{\partial^2 S_a}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb}\left(-S_d\, \Gamma^d{}_{ea} + \frac{\partial S_a}{\partial x^e}\right) - \Gamma^d{}_{ba}\, \frac{\partial S_d}{\partial x^c} - \Gamma^e{}_{ca}\left(-S_d\, \Gamma^d{}_{be} + \frac{\partial S_e}{\partial x^b}\right) - S_d\, \frac{\partial \Gamma^d{}_{ba}}{\partial x^c}\right)$$

$$\text{Out[76]=}\quad -S_d\, S^e\, \Gamma^a{}_{ce}\, \Gamma^d{}_{ba} + S_d\, S^a\, \Gamma^d{}_{be}\, \Gamma^e{}_{ca} + \frac{\partial^2 \phi}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb}\, \frac{\partial \phi}{\partial x^e} +$$
$$S^a\, \frac{\partial^2 S_a}{\partial x^c\, \partial x^b} - S^a\, \Gamma^e{}_{cb}\, \frac{\partial S_a}{\partial x^e} + S_a\, \frac{\partial^2 S^a}{\partial x^c\, \partial x^b} + \frac{\partial S_a}{\partial x^c}\, \frac{\partial S^a}{\partial x^b} + \frac{\partial S_a}{\partial x^b}\, \frac{\partial S^a}{\partial x^c} - S_a\, \Gamma^e{}_{cb}\, \frac{\partial S^a}{\partial x^e}$$

$$\text{Out[77]=}\quad -S_d\, S^e\, \Gamma^a{}_{ce}\, \Gamma^d{}_{ba} + S_d\, S^a\, \Gamma^d{}_{be}\, \Gamma^e{}_{ca} + \frac{\partial^2 \phi}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb}\, \frac{\partial \phi}{\partial x^e} +$$
$$S^a\, \frac{\partial^2 S_a}{\partial x^c\, \partial x^b} - S^a\, \Gamma^e{}_{cb}\, \frac{\partial S_a}{\partial x^e} + S_a\, \frac{\partial^2 S^a}{\partial x^c\, \partial x^b} + \frac{\partial S_a}{\partial x^c}\, \frac{\partial S^a}{\partial x^b} + \frac{\partial S_a}{\partial x^b}\, \frac{\partial S^a}{\partial x^c} - S_a\, \Gamma^e{}_{cb}\, \frac{\partial S^a}{\partial x^e}$$

We cannot expand the following covariant derivatives because they contain non tensor items.

```
In[78]:= {CovariantD2d[PartialD[labs0][Td[a], xu[b]], c], CovariantD2d[Γudd[a, b, c], d]}
         % // ExpandCovariantD2d[labs0, i]
```

$$\text{Out[78]=}\quad \left\{\left(\frac{\partial T_a}{\partial x^b}\right)_{|\,c},\ \Gamma^a{}_{bc\,|\,d}\right\}$$

```
ExpandCovariantD::nottensor :
 A covariant derivative , {(∂T_a/∂x^b)_;c, Γ^a_bc;d}, cannot be expanded
   because Tensorial cannot assess the tensor nature of the expression.
```

```
Out[79]= $Aborted
```

However, by setting permissive to True, Tensorial will do the expansion. (It would be incorrect in this example.)

```
In[80]:= {CovariantD2d[PartialD[labs0][Td[a], xu[b]], red@c], CovariantD2d[Γudd[a, b, c], d]} //
          ToFlavorD[red]
         % // ExpandCovariantD2d[labs0, red@i, True]
```

$$\text{Out[80]=}\quad \left\{\left(\frac{\partial T_a}{\partial x^b}\right)_{|\,c},\ \Gamma^a{}_{bc\,|\,d}\right\}$$

$$\text{Out[81]=}\quad \left\{\frac{\partial^2 T_a}{\partial x^c\, \partial x^b} - \Gamma^i{}_{cb}\, \frac{\partial T_a}{\partial x^i} - \Gamma^i{}_{ca}\, \frac{\partial T_i}{\partial x^b},\ \Gamma^a{}_{di}\, \Gamma^i{}_{bc} - \Gamma^a{}_{ic}\, \Gamma^i{}_{db} - \Gamma^a{}_{bi}\, \Gamma^i{}_{dc} + \frac{\partial \Gamma^a{}_{bc}}{\partial x^d}\right\}$$

The following expression won't expand at all because it contains a base index. (Take the covariant derivative of the abstract index expression and then expand to base indices.)

```
In[82]:= CovariantD2d[Td[1], b]
         % // ExpandCovariantD2d[labs0, i, True]
```

$$\text{Out[82]=}\quad T_{1\,|\,b}$$

```
ExpandCovariantD::nottensor :
 A covariant derivative , T_1;b, cannot be expanded because
   Tensorial cannot assess the tensor nature of the expression.
```

```
Out[83]= $Aborted
```

The following expression won't expand because it contains an unexpanded partial derivative.

```
In[84]:= PartialD[Td[a], b]
         CovariantD2d[%, c]
         % // ExpandCovariantD2d[labs0, i, True]
```

Out[84]= $T_{a,b}$

Out[85]= $(T_{a,b})_{|c}$

```
       ExpandCovariantD::nottensor :
        A covariant derivative , (T_{a,b})_{;c}, cannot be expanded because
          Tensorial cannot assess the tensor nature of the expression.
```

```
Out[86]= $Aborted
```

Expand the partial derivative first.

```
In[87]:= PartialD[Td[a], b]
         CovariantD2d[%, c]
         % // ExpandPartialD[labs0]
         % // ExpandCovariantD2d[labs0, i, True]
```

Out[87]= $T_{a,b}$

Out[88]= $(T_{a,b})_{|c}$

Out[89]= $\left(\frac{\partial T_a}{\partial x^b}\right)_{|c}$

Out[90]= $\frac{\partial^2 T_a}{\partial x^c \partial x^b} - \Gamma^i{}_{cb} \frac{\partial T_a}{\partial x^i} - \Gamma^i{}_{ca} \frac{\partial T_i}{\partial x^b}$

The following is a second order partial derivative of a tensor.

```
In[91]:= Td[a]
         CovariantD2d[%, {b, c}]
         step1 = % // ExpandCovariantD2d[labs0, {i, j}]
```

Out[91]= T_a

Out[92]= $T_{a|bc}$

Out[93]= $\frac{\partial^2 T_a}{\partial x^c \partial x^b} - \Gamma^j{}_{cb} \left(-T_i \Gamma^i{}_{ja} + \frac{\partial T_a}{\partial x^j}\right) - \Gamma^i{}_{ba} \frac{\partial T_i}{\partial x^c} - \Gamma^j{}_{ca} \left(-T_i \Gamma^i{}_{bj} + \frac{\partial T_j}{\partial x^b}\right) - T_i \frac{\partial \Gamma^i{}_{ba}}{\partial x^c}$

The following steps perform the covariant derivative in two separate steps. The first step generates partial derivatives and Christoffels. Nevertheless, the complete expression is a valid tensor so the user would be justified in granting permission to covariantly expand.

```
In[94]:= Td[a]
         CovariantD2d[%, b]
         % // ExpandCovariantD2d[labs0, i]
         Print[
          "The expression above is a valid tensor even though the individual terms aren't"]
         CovariantD2d[%%, c]
         step2 = % // ExpandCovariantD2d[labs0, j, True]
         Print["Check that the two answers are the same"]
         step1 - step2 // TensorSimplify
```

Out[94]= T_a

Out[95]= $T_{a|b}$

Out[96]= $-T_i \Gamma^i{}_{ba} + \frac{\partial T_a}{\partial x^b}$

```
          The expression above is a valid tensor even though the individual terms aren't
```

Out[98]= $\left(\frac{\partial T_a}{\partial x^b}\right)_{|c} - \Gamma^i{}_{ba|c} T_i - T_{i|c} \Gamma^i{}_{ba}$

Out[99]= $\frac{\partial^2 T_a}{\partial x^c \partial x^b} - \Gamma^j{}_{cb} \frac{\partial T_a}{\partial x^j} - \Gamma^i{}_{ba} \left(-T_j \Gamma^j{}_{ci} + \frac{\partial T_i}{\partial x^c}\right) -$

$\Gamma^j{}_{ca} \frac{\partial T_j}{\partial x^b} - T_i \left(\Gamma^i{}_{cj} \Gamma^j{}_{ba} - \Gamma^i{}_{bj} \Gamma^j{}_{ca} - \Gamma^i{}_{ja} \Gamma^j{}_{cb} + \frac{\partial \Gamma^i{}_{ba}}{\partial x^c}\right)$

```
          Check that the two answers are the same
```

```
Out[101]= 0
```

Restore settings.

```
In[102]:= ClearTensorValues /@
            {xu[i], add[i, j], auu[i, j], Γudd[i, j, k], Γddd[i, j, k], δud[i, j]};

In[103]:= ClearTensorShortcuts[{{x, y, xb, S, T}, 1}, {{a, g, δ, S, T}, 2}, {{Γ, Γb, Λ, zero}, 3}]

In[104]:= DeclareBaseIndices @@ oldindices
          ClearIndexFlavor /@ IndexFlavors;
          DeclareIndexFlavor /@ oldflavors;
          Clear[oldindices, oldflavors, labs0, labsz, step1, step2, covmn]
```

FullLeviCivitaExpand

■ FullLeviCivitaExpand[*e*, *g*][*expr*] will fully expand product of pairs of LeviCivita tensors with label e, as metric tensors g.

■ FullLeviCivitaExpand also applies in dimension two.

■ Labels other than *e* can be used to represent the Levi Civita tensors.

■ See also: LeviCivitaOrder, LeviCivitaSimplify, ContractKroneckers, FullKroneckerExpand.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]

In[3]:= oldflavors = IndexFlavors;
        DeclareIndexFlavor[{red, Red}];

In[5]:= DefineTensorShortcuts[{{g, e}, 2}, {e, 3}]
```

The following routine fully expands product of Levi Civita tensors.

```
In[6]:= euuu[i, j, k] eddd[i, m, n]
        % // FullLeviCivitaExpand[e, g]
        euuu[i, j, k] eddd[i, m, n] == (% /. {gdu[δ_, δ_] → NDim, gud[δ_, δ_] → NDim})
```

$$\text{Out[6]}= e_{imn}\, e^{ijk}$$

$$\text{Out[7]}= -(g_i{}^i - 2)\,(g_n{}^j\, g_m{}^k - g_m{}^j\, g_n{}^k)$$

$$\text{Out[8]}= e_{imn}\, e^{ijk} = g_m{}^j\, g_n{}^k - g_n{}^j\, g_m{}^k$$

It is easy to show that it applies also in dimension two ($\alpha, \beta, \delta, \gamma = \{1,2\}$):

```
In[9]:= euu[red@β, red@δ] edd[red@α, red@γ] ==
          (euu[red@β, red@δ] edd[red@α, red@γ] // FullLeviCivitaExpand[e, g])
```

$$\text{Out[9]}= e_{\alpha\gamma}\, e^{\beta\delta} = g_\alpha{}^\beta\, g_\gamma{}^\delta - g_\gamma{}^\beta\, g_\alpha{}^\delta$$

```
In[10]:= ClearTensorShortcuts[{{g, e}, 2}, {e, 3}]
```

This resets to the old indices.

```
In[11]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

FunctionalD

■ FunctionalD[*g*, *flv*][L, f] calculates the functional derivative of the functional form L[flv[k], f, $f_{\text{flv[k]}}$, ...] with respect to f and its derivatives. g is the metric tensor in a basis with flavor flv.

FunctionalD[*g*, *flv*, *index1*, *index2*][L, f] introduces additional indices index1,index2 in the functional derivative of the functional form with respect to f . This is necessary when L contains covariant or contravariant derivatives.

■ `index1,index2` must be chosen not to interfere with the existing indices.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[6]:= DefineTensorShortcuts[{g, 2}]

In[7]:= f = Tensor[Q, {red@p, Void}, {Void, red@q}]
```

$$\text{Out[7]}= Q^{p}{}_{q}$$

```
In[8]:= Tensor[Q, {red@i, Void}, {Void, red@i}]
        δ[%] / δ[Tensor[Q, {red@p, Void}, {Void, red@q}]] == FunctionalD[g, red][%, f]
```

$$\text{Out[8]}= Q^{i}{}_{i}$$

$$\text{Out[9]}= \frac{\delta(Q^{i}{}_{i})}{\delta(Q^{p}{}_{q})} = g^{q}{}_{p}$$

```
In[10]:= Tensor[Q, {red@i, Void}, {Void, red@j}] Tensor[Q, {red@j, Void}, {Void, red@k}]
         δ[%] / δ[Tensor[Q, {red@p, Void}, {Void, red@q}]] == FunctionalD[g, red][%, f]
```

$$\text{Out[10]}= Q^{i}{}_{j}\, Q^{j}{}_{k}$$

$$\text{Out[11]}= \frac{\delta(Q^{i}{}_{j}\, Q^{j}{}_{k})}{\delta(Q^{p}{}_{q})} = g^{q}{}_{k}\, Q^{i}{}_{p} + g^{i}{}_{p}\, Q^{q}{}_{k}$$

```
In[12]:= Tensor[Q, {red@i, Void}, {Void, red@j}] Tensor[P, {red@j, Void}, {Void, red@k}]
         δ[%] / δ[Tensor[Q, {red@p, Void}, {Void, red@q}]] == FunctionalD[g, red][%, f]
```

$$\text{Out[12]}= P^{i}{}_{k}\, Q^{i}{}_{j}$$

$$\text{Out[13]}= \frac{\delta(P^{i}{}_{k}\, Q^{i}{}_{j})}{\delta(Q^{p}{}_{q})} = g^{i}{}_{p}\, P^{q}{}_{k}$$

```
In[14]:= Tensor[Q, {red@i, red@j}, {Void, Void}] Tensor[Q, {Void, Void}, {red@h, red@k}]
         δ[%] / δ[Tensor[Q, {red@p, Void}, {Void, red@q}]] == FunctionalD[g, red][%, f]
```

$$\text{Out[14]}= Q_{h\,k}\, Q^{i\,j}$$

$$\text{Out[15]}= \frac{\delta(Q_{h\,k}\, Q^{i\,j})}{\delta(Q^{p}{}_{q})} = g^{i}{}_{p}\, g^{q\,j}\, Q_{h\,k} + g_{p\,h}\, g^{q}{}_{k}\, Q^{i\,j}$$

The following case shows that additional indicies are needed :

```
In[16]:= CovariantD[Tensor[Q, {red@i, red@j}, {Void, Void}], red@k]
          ContravariantD[Tensor[Q, {Void, Void}, {red@i, red@j}], red@k]
         δ[%] / δ[Tensor[Q, {red@p, Void}, {Void, red@q}]] == FunctionalD[g, red][%, f]
```

$$\text{Out[16]}= Q^{i\,j}{}_{;k}\, Q_{i\,j}{}^{;k}$$

$$\text{Out[17]}= \frac{\delta\left(Q^{i\,j}{}_{;k}\, Q_{i\,j}{}^{;k}\right)}{\delta(Q^{p}{}_{q})} = -2\,(Q_{p}{}^{q;\text{TContinuumMechanics2`TContinuumMechanics`Private`r\$}})_{;\text{TContinuumMechanics2`TContinuumMechanics`Private`r\$}}$$

this is realized by including n and m indicies (in the present case only index n is used) :

```
In[18]:= CovariantD[Tensor[Q, {red@i, red@j}, {Void, Void}], red@k]
          ContravariantD[Tensor[Q, {Void, Void}, {red@i, red@j}], red@k]
         δ[%] / δ[Tensor[Q, {red@p, Void}, {Void, red@q}]] == FunctionalD[g, red, n, m][%, f]
```

$$\text{Out[18]}= Q^{i\,j}{}_{;k}\, Q_{i\,j}{}^{;k}$$

$$\text{Out[19]}= \frac{\delta\left(Q^{i\,j}{}_{;k}\, Q_{i\,j}{}^{;k}\right)}{\delta(Q^{p}{}_{q})} = -2\,(Q_{p}{}^{q;n})_{;n}$$

This expression is nothing but -2 times the laplacian of $Q_{p}{}^{q}$

```
In[20]:= -2 TLaplacian[e, g, red@n, red@m][Tensor[Q, {Void, red@q}, {red@p, Void}]]
         % // MetricSimplifyD[g]
```

$$\text{Out[20]}= -2\, Q_p{}^{q}{}_{;m\,n}\, g^{m\,n}$$

$$\text{Out[21]}= -2\, (Q_p{}^{q\,;n})_{;n}$$

```
In[22]:= ClearTensorShortcuts[{{g}, 2}]
```

```
In[23]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

HoldContravariantD2d

- HoldContravariantD2d[*expr*, *i*] represents the unexpanded contravariant derivative of expr with respect to the index i.
- HoldContravariantD2d[*expr*, {*i*, *j*, ...}] represents the unexpanded contravariant derivative of the expression with respect to the list of indices.

- Contravariant derivatives are expanded as does Covariant derivatives. HoldContravariantD2d must be used if we want to leave the 2-dimensional contravariant derivative of an expression unexpanded (Hold Form)
- ReleaseHoldD is used to expand the any contravariant derivatives (and not ReleaseHold !).
- The intended flavors must be on the indices in HoldContravariantD2d.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]
```

```
In[7]:= DefineTensorShortcuts[{{R, S, T}, 1}, {{R, S, T}, 2}]
```

Here is a tensor T and its contravariant derivative with respect to index k. In the FullForm we see that the contravariant index is written in an Hold Form.

```
In[8]:= Td[i]
        ContravariantD2d[%, k]
        % // FullForm
```

$$\text{Out[8]}= T_i$$

$$\text{Out[9]}= T_i{}^{|k}$$

```
Out[10]//FullForm=
HoldContravariantD2d[Tensor[T, List[Void], List[black[i]]], black[k]]
```

As normal linear and Liebnizian differentiation rules are applied in contravariant derivations, we need HoldContravariantD2d to write the expression in an hold form:

```
In[11]:= {Su[i] + Tu[i], Su[i] Tu[j]} // TableForm
         HoldContravariantD2d[#, {h, k}] & /@ % // TableForm
```

```
Out[11]//TableForm=
S^i + T^i
S^i T^j
```

```
Out[12]//TableForm=
(S^i + T^i)|h k
(S^i T^j)|h k
```

The flavor must be on the contravariant index when differentiating.

ReleaseHoldD is used to expand the contravariant derivatives (and not ReleaseHold !).

```
In[13]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         HoldContravariantD2d[%, red@k]
         % // ReleaseHoldD
```

$$\text{Out[13]}= \frac{2\,a\,\pi\,S^{i}\,T^{j}}{\nu+1}$$

$$\text{Out[14]}= \left(\frac{2\,a\,\pi\,S^{i}\,T^{j}}{\nu+1}\right)^{|k}$$

$$\text{Out[15]}= \frac{2\,a\,\pi\left(T^{j|k}\,S^{i}+S^{i|k}\,T^{j}\right)}{\nu+1}$$

Restore settings.

```
In[16]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

```
In[17]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

HoldContravariantD

- `HoldContravariantD[expr, i]` represents the unexpanded contravariant derivative of expr with respect to the index `i`.
- `HoldContravariantD[expr, {i, j, ...}]` represents the unexpanded contravariant derivative of the expression with respect to the list of indices.

- Contravariant derivatives are expanded as does Covariant derivatives. HoldContravariantD must be used if we want to leave the expression unexpanded (Hold Form)
- `ReleaseHoldD` is used to expand the contravariant derivatives (and not `ReleaseHold` !).
- The intended flavors must be on the indices in Hold`ContravariantD`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]
```

```
In[7]:= DefineTensorShortcuts[{{R, S, T}, 1}, {{R, S, T}, 2}]
```

Here is a tensor T and its contravariant derivative with respect to index k. In the FullForm we see that the contravariant index is written in an Hold Form.

```
In[8]:= Td[i]
        ContravariantD[%, k]
        % // FullForm
```

$$\text{Out[8]}= T_{i}$$

$$\text{Out[9]}= T_{i}^{;k}$$

```
Out[10]//FullForm=
HoldContravariantD[Tensor[T, List[Void], List[black[i]]], black[k]]
```

As normal linear and Liebnizian differentiation rules are applied in contravariant derivations, we need **HoldContravariantD** to write the expression in an hold form:

```
In[11]:= {Su[i] + Tu[i], Su[i] Tu[j]} // TableForm
         HoldContravariantD[#, {h, k}] & /@ % // TableForm
```

Out[11]//TableForm=

$$S^i + T^i$$

$$S^i\, T^j$$

Out[12]//TableForm=

$$(S^i + T^i)^{;h\,k}$$

$$(S^i\, T^j)^{;h\,k}$$

The flavor must be on the contravariant index when differentiating.

ReleaseHoldD is used to expand the contravariant derivatives (and not ReleaseHold !).

```
In[13]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         HoldContravariantD[%, red@k]
         % // ReleaseHoldD
```

Out[13]=

$$\frac{2\,a\,\pi\,S^i\,T^j}{\nu+1}$$

Out[14]=

$$\left(\frac{2\,a\,\pi\,S^i\,T^j}{\nu+1}\right)^{;k}$$

Out[15]=

$$\frac{2\,a\,\pi\left(T^{j\,;k}\,S^i + S^{i\,;k}\,T^j\right)}{\nu+1}$$

Restore settings.

```
In[16]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

```
In[17]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

HoldCovariantD2d

- HoldCovariantD2d[*expr*, *i*] represents the unexpanded 2-dimensional covariant derivative of expr with respect to the index i.
- HoldCovariantD2d[*expr*, {*i*, *j*, ...}] represents the unexpanded 2-dimensional covariant derivative of the expression with respect to the list of indices.

- Because two-dimensional covariant derivatives are expanded as does Covariant derivatives, HoldCovariantD2d must be used if we want to leave the expression unexpanded (Hold Form)
- ReleaseHold is used to expand the derivatives.
- The intended flavors must be on the indices in HoldCovariantD2d.
- See also: ExpandCovariantD2d, CovariantD2d,

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]
```

```
In[7]:= DefineTensorShortcuts[{{R, S, T}, 1}, {{R, S, T}, 2}]
```

Here is a tensor T and its 2-d covariant derivative with respect to index k (by default a "|" is used). In the FullForm we see that the 2d covariant index is written in an Hold Form (this is not the case for CovariantD, the general covariant derivatives).

```
In[8]:= Td[i]
        CovariantD2d[%, k]
        % // FullForm
```

Out[8]= T_i

Out[9]= $T_{i\,|\,k}$

```
Out[10]//FullForm=
HoldCovariantD2d[Tensor[T, List[Void], List[black[i]]], black[k]]
```

As normal linear and Liebnizian differentiation rules are applied in covariant derivations, we need **HoldCovariantD2d** to write the expression in an hold form:

```
In[11]:= {Su[i] + Tu[i], Su[i] Tu[j]} // TableForm
         HoldCovariantD2d[#, {h, k}] & /@ % // TableForm
```

```
Out[11]//TableForm=
S^i + T^i
S^i T^j
```

```
Out[12]//TableForm=
(S^i + T^i)_|hk
(S^i T^j)_|hk
```

The flavor must be on the covariant index when differentiating.

`ReleaseHold` is used to expand the derivatives.

```
In[13]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         HoldCovariantD2d[%, red@k]
         % // ReleaseHold
```

Out[13]= $\frac{2\,a\,\pi\,S^i\,T^j}{\nu+1}$

Out[14]= $\left(\frac{2\,a\,\pi\,S^i\,T^j}{\nu+1}\right)_{|\,k}$

Out[15]= $\frac{2\,a\,\pi\left(T^j{}_{|\,k}\,S^i + S^i{}_{|\,k}\,T^j\right)}{\nu+1}$

Restore settings.

```
In[16]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

```
In[17]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

HoldCovariantD

- `HoldCovariantD[`*expr*, *i*`]` represents the unexpanded covariant derivative of expr with respect to the index `i`.
- `HoldCovariantD[`*expr*, {*i*, *j*, ...}`]` represents the unexpanded covariant derivative of the expression with respect to the list of indices.

- Covariant derivatives are expanded by default. In *Tensorial* 4, the operator `HoldOp` allows to conserve the expressions in a hold form. For homogeneity reasons, we have define in *TContinuumMechanics* 2 `HoldCovariantD`. `HoldCovariantD` must be used if we want to leave the expression unexpanded (Hold Form)
- `ReleaseHold` is used to expand the derivatives.
- The intended flavors must be on the indices in `HoldCovariantD`.
- See also: `ExpandCovariantD`, `CovariantD`, `HoldOp`,

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]

In[7]:= DefineTensorShortcuts[{{R, S, T}, 1}, {{R, S, T}, 2}]
```

As normal linear and Liebnizian differentiation rules are applied in covariant derivations, we need either **HoldOp** or **HoldCovariantD** to write the expression in an hold form:

```
In[8]:= {Su[i] + Tu[i], Su[i] Tu[j]} // TableForm
        HoldCovariantD[#, {h, k}] & /@ % // TableForm
```

```
Out[8]//TableForm=
```

$$S^i + T^i$$

$$S^i\, T^j$$

```
Out[9]//TableForm=
```

$$(S^i + T^i)_{;h\,k}$$

$$(S^i\, T^j)_{;h\,k}$$

The flavor must be on the covariant index when differentiating.

`ReleaseHold` is used to expand the derivatives.

```
In[10]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         HoldCovariantD[%, red@k]
         % // ReleaseHold
```

$$\text{Out[10]=}\quad \frac{2\,a\,\pi\,S^i\,T^j}{\nu+1}$$

$$\text{Out[11]=}\quad \left(\frac{2\,a\,\pi\,S^i\,T^j}{\nu+1}\right)_{;k}$$

$$\text{Out[12]=}\quad \frac{2\,a\,\pi\left(T^j{}_{;k}\,S^i + S^i{}_{;k}\,T^j\right)}{\nu+1}$$

Restore settings.

```
In[13]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]

In[14]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

Kronecker

■ `Kronecker[δ][expr]` gives the Kronecker delta δ in expr, equal to 1 if the indices with the same flavor are equal, and 0 otherwise.

■ See also: `KroneckerAbsorb`, `KroneckerDelta`, `KroneckerProduct`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[6]:= DefineTensorShortcuts[{g, 2}]
```

```
In[7]:= h gud[i, j]
        % // ArrayExpansion[{i, j}]
        % // Kronecker[g]
```

Out[7]= $h\,g^{i}{}_{j}$

Out[8]= $h\,g^{i}{}_{j}$

Out[9]= $h\,g^{i}{}_{j}$

```
In[10]:= h gud[red@i, red@j]
         % // ArrayExpansion[{red@i, red@j}]
         % // Kronecker[g]
```

Out[10]= $h\,g^{i}{}_{j}$

Out[11]= $\begin{pmatrix} h\,g^{1}{}_{1} & h\,g^{1}{}_{2} & h\,g^{1}{}_{3} \\ h\,g^{2}{}_{1} & h\,g^{2}{}_{2} & h\,g^{2}{}_{3} \\ h\,g^{3}{}_{1} & h\,g^{3}{}_{2} & h\,g^{3}{}_{3} \end{pmatrix}$

Out[12]= $\begin{pmatrix} h & 0 & 0 \\ 0 & h & 0 \\ 0 & 0 & h \end{pmatrix}$

Kronecker does not act on indices with different flavors,

```
In[13]:= h gud[red@i, j]
         % // ArrayExpansion[{red@i, black@j}]
         % // Kronecker[g]
```

Out[13]= $h\,g^{i}{}_{j}$

Out[14]= $\begin{pmatrix} h\,g^{1}{}_{1} & h\,g^{1}{}_{2} & h\,g^{1}{}_{3} \\ h\,g^{2}{}_{1} & h\,g^{2}{}_{2} & h\,g^{2}{}_{3} \\ h\,g^{3}{}_{1} & h\,g^{3}{}_{2} & h\,g^{3}{}_{3} \end{pmatrix}$

Out[15]= $\begin{pmatrix} h\,g^{1}{}_{1} & h\,g^{1}{}_{2} & h\,g^{1}{}_{3} \\ h\,g^{2}{}_{1} & h\,g^{2}{}_{2} & h\,g^{2}{}_{3} \\ h\,g^{3}{}_{1} & h\,g^{3}{}_{2} & h\,g^{3}{}_{3} \end{pmatrix}$

```
In[16]:= ClearTensorShortcuts[{g, 2}]
```

```
In[17]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

KroneckerAbsorbD

■ `KroneckerAbsorbD[δ][expr]` extends the operation KroneckerAbsorb[δ] to hold forms of covariant derivations (HoldCovariantD, HoldCovariantD2d,...).

■ Labels other that δ can be used to represent the Kronecker.

■ See also: `FullKroneckerExpand`, `PartialKroneckerExpand`, `KroneckerContract`, `MapLevelParts`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings and declare base indices and flavors.

```
In[3]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareBaseIndices[{1, 2, 3}]
        DeclareIndexFlavor /@ {{red, Red}};
```

```
In[8]:= HoldCovariantD[b^δ_γ u_δ, red[α]] a^γ_β
        % // KroneckerAbsorb[a]
        %% // KroneckerAbsorbD[a]
```

Out[8]= $(b^{\delta}{}_{\gamma}\, u_{\delta})_{;\alpha}\, a^{\gamma}{}_{\beta}$

Out[9]= $(b^{\delta}{}_{\gamma}\, u_{\delta})_{;\alpha}\, a^{\gamma}{}_{\beta}$

Out[10]= $(b^{\delta}{}_{\beta}\, u_{\delta})_{;\alpha}$

```
In[11]:= HoldCovariantD[b^δ_γ u_δ, red[α]] ( a^γ_β + a_β^γ )
         KroneckerAbsorbD[a] /@ Expand@%
```

Out[11]= $(b^{\delta}{}_{\gamma}\, u_{\delta})_{;\alpha}\, (a_{\beta}{}^{\gamma} + a^{\gamma}{}_{\beta})$

Out[12]= $2\, (b^{\delta}{}_{\beta}\, u_{\delta})_{;\alpha}$

```
In[13]:= HoldCovariantD[b^δ_γ u_δ, red[α]] (a_β^α + a^α_β)
         KroneckerAbsorbD[a] /@ Expand@%
```

Out[13]= $(b^{\delta}{}_{\gamma}\, u_{\delta})_{;\alpha}\, (a_{\beta}{}^{\alpha} + a^{\alpha}{}_{\beta})$

Out[14]= $2\, (b^{\delta}{}_{\gamma}\, u_{\delta})_{;\beta}$

```
In[15]:= HoldCovariantD[b^δ_γ u_δ, red[α]] a^γ_β + b_αγ a^γ_β
         KroneckerAbsorbD[a] /@ %
```

Out[15]= $(b^{\delta}{}_{\gamma}\, u_{\delta})_{;\alpha}\, a^{\gamma}{}_{\beta} + a^{\gamma}{}_{\beta}\, b_{\alpha\gamma}$

Out[16]= $(b^{\delta}{}_{\beta}\, u_{\delta})_{;\alpha} + b_{\alpha\beta}$

```
In[17]:= HoldCovariantD2d[b^δ_γ u_δ, red[α]] a^γ_β + b_αγ a^γ_β
         KroneckerAbsorbD[a] /@ %
```

Out[17]= $(b^{\delta}{}_{\gamma}\, u_{\delta})_{|\alpha}\, a^{\gamma}{}_{\beta} + a^{\gamma}{}_{\beta}\, b_{\alpha\gamma}$

Out[18]= $(b^{\delta}{}_{\beta}\, u_{\delta})_{|\alpha} + b_{\alpha\beta}$

```
In[19]:= HoldContravariantD[b^δ_γ u_δ, red[β]] a^γ_β
         % // KroneckerAbsorbD[a]
```

Out[19]= $(b^{\delta}{}_{\gamma}\, u_{\delta})^{;\beta}\, a^{\gamma}{}_{\beta}$

Out[20]= $(b^{\delta}{}_{\gamma}\, u_{\delta})^{;\gamma}$

```
In[23]:= HoldContravariantD2d[b^δ_γ u_δ, red[β]] a^γ_β + b^β_γ a^γ_β
         KroneckerAbsorbD[a] /@ %
```

Out[23]= $(b^{\delta}{}_{\gamma}\, u_{\delta})^{|\beta}\, a^{\gamma}{}_{\beta} + a^{\gamma}{}_{\beta}\, b^{\beta}{}_{\gamma}$

Out[24]= $(b^{\delta}{}_{\gamma}\, u_{\delta})^{|\gamma} + b^{\beta}{}_{\beta}$

```
In[25]:= HoldContravariantD[b^δ_γ u_δ, red[α]] a^γ_β
         KroneckerAbsorbD[a]@%
```

Out[25]= $(b^{\delta}{}_{\gamma}\, u_{\delta})^{;\alpha}\, a^{\gamma}{}_{\beta}$

Out[26]= $(b^{\delta}{}_{\beta}\, u_{\delta})^{;\alpha}$

```
In[27]:= HoldContravariantD2d[b^δ_γ u_δ, red[α]] a^γ_β
         KroneckerAbsorbD[a]@%
```

Out[27]= $(b^{\delta}{}_{\gamma}\, u_{\delta})^{|\alpha}\, a^{\gamma}{}_{\beta}$

Out[28]= $(b^{\delta}{}_{\beta}\, u_{\delta})^{|\alpha}$

```
In[29]:= HoldContravariantD[b^δ_γ, red[α]] a^γ_β
         KroneckerAbsorbD[a]@%
         HoldContravariantD[b^δ_γ, red[α]] a^γ_α
         KroneckerAbsorbD[a]@%
```

Out[29]= $b^{\delta}{}_{\gamma}{}^{;\alpha}\, a^{\gamma}{}_{\beta}$

Out[30]= $b^{\delta}{}_{\beta}{}^{;\alpha}$

Out[31]= $b^{\delta}{}_{\gamma}{}^{;\alpha}\, a^{\gamma}{}_{\alpha}$

Out[32]= $b^{\delta}{}_{\gamma}{}^{;\gamma}$

```
In[35]:= ContravariantD2d[b^δ_γ u_δ, red[β]] a^γ_β + b^β_γ a^γ_β
         KroneckerAbsorbD[a] /@ Expand@%
```

Out[35]= $a^{\gamma}{}_{\beta}\, b^{\beta}{}_{\gamma} + a^{\gamma}{}_{\beta}\, \left(u_{\delta}{}^{|\beta}\, b^{\delta}{}_{\gamma} + b^{\delta}{}_{\gamma}{}^{|\beta}\, u_{\delta}\right)$

Out[36]= $b^{\beta}{}_{\beta} + u_{\delta}{}^{|\beta}\, b^{\delta}{}_{\beta} + b^{\delta}{}_{\gamma}{}^{|\beta}\, a^{\gamma}{}_{\beta}\, u_{\delta}$

```
In[33]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

TLaplaceOp

■ `TLaplaceOp[Δ][expr]` writes all the laplacians in expr, using a Δ notation.

■ `LaplaceOp` applies on metric simplified expressions.

■ See also: `TLaplacian`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]
        DefineTensorShortcuts[{T, 1}, {T, 2}]
```

```
In[7]:= Tf := Tensor[f]
        TLaplacian[e, g, red@i][Tf]
        % // LaplaceOp[Δ]
```

Out[8]= $(f^{i})_{;i}$

Out[9]= $\Delta(f)$

`LaplaceOp` applies on metric simplified expressions :

```
In[10]:= Tf := Tensor[f]
         TLaplacian[e, g, red@j, red@i][Tf]
         % // LaplaceOp[Δ]
         % // MetricSimplifyD[g]
         % // LaplaceOp[Δ]
```

Out[11]= $f_{;i\,j}\, g^{i\,j}$

Out[12]= $f_{;i\,j}\, g^{i\,j}$

Out[13]= $(f^{j})_{;j}$

Out[14]= $\Delta(f)$

In the case of Laplacian of non scalar tensor, `LaplaceOp` gives :

```
In[15]:= TLaplacian[e, g, red@i][Tu[red@j]]
         % // LaplaceOp[Δ]
```

Out[15]= $\left(T^{j\,;i}\right)_{;i}$

Out[16]= $\Delta(T)^{j}$

Restore the settings.

```
In[17]:= ClearTensorShortcuts[{T, 1}, {T, 2}]
```

```
In[18]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

LeviCivitaOrder

■ `LeviCivitaOrder[e][expr]` orders the indices in the Levi-Civita symbol 'e', taking into account the signature of the permutation, and assign to zero its covariant derivatives.

■ See also: `CovariantD`, `LeviCivitaSimplify`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]
        labels = {x, δ, g, Γ};
```

```
In[7]:= DefineTensorShortcuts[{{a, b, e}, 1}, {e, 3}]
```

```
In[8]:= a = au[red@i] ed[red@i]
        b = bu[red@j] ed[red@j]
        a × b // LinearBreakout[Cross][ed[_]]
        (% // CrossProductExpansion[e, e, k]) // LeviCivitaOrder[e]
```

Out[8]= $a^{i}\,\mathbf{e}_{i}$

Out[9]= $b^{j}\,\mathbf{e}_{j}$

Out[10]= $\mathbf{e}_{i}\times\mathbf{e}_{j}\,a^{i}\,b^{j}$

Out[11]= $a^{i}\,b^{j}\,e_{i\,j\,k}\,\mathbf{e}^{k}$

It assign to zero the covariant derivatives of the Levi-Civita symbol :

```
In[12]:= CovariantD[%, red@h]
         % // LeviCivitaOrder[e]
```

Out[12]= $a^{i}{}_{;h}\,b^{j}\,e_{i\,j\,k}\,\mathbf{e}^{k}+a^{i}\left(b^{j}{}_{;h}\,e_{i\,j\,k}\,\mathbf{e}^{k}+b^{j}\left(\mathbf{e}^{k}{}_{;h}\,e_{i\,j\,k}+e_{i\,j\,k\,;h}\,\mathbf{e}^{k}\right)\right)$

Out[13]= $a^{i}{}_{;h}\,b^{j}\,e_{i\,j\,k}\,\mathbf{e}^{k}+a^{i}\left(b^{j}{}_{;h}\,e_{i\,j\,k}\,\mathbf{e}^{k}+b^{j}\left(\mathbf{e}^{k}{}_{;h}\,e_{i\,j\,k}+e_{i\,j\,k\,;h}\,\mathbf{e}^{k}\right)\right)$

```
In[14]:= % // CovariantDSimplify[e, g, e]
```

Out[14]= $b^{j}{}_{;h}\,a^{i}\,e_{i\,j\,k}\,\mathbf{e}^{k}+a^{i}{}_{;h}\,b^{j}\,e_{i\,j\,k}\,\mathbf{e}^{k}$

but not the partial derivatives $\frac{\partial e_{123}}{\partial x^{j}}$:

```
In[15]:= CovariantD[Tensor[e, {Void, Void, Void}, {red@1, red@2, red@3}], red@l]
         % // ExpandCovariantD[labels, red@h] // SumExpansion[red@h] // FullSimplify
         (% // LeviCivitaOrder[e]) == 0
```

Out[15]= $e_{1\,2\,3\,;l}$

ExpandCovariantD::nottensor :
A covariant derivative , $e_{1\,2\,3\,;l}$, cannot be expanded because *Tensorial* cannot assess the tensor nature of the expression.

Out[16]= $Aborted

Out[17]= $Aborted = 0

Restore the settings.

```
In[18]:= ClearTensorShortcuts[{{a, b, e}, 1}, {e, 3}]
```

```
In[19]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

LeviCivitaSimplify

- `LeviCivitaSimplify[e, ϵ][g, flavor][expr]` transforms in *expr* the LeviCivita tensors *e* with down (resp. up) indices into the usual permutation symbol ϵ multiplied (resp. divided) by the square root on the elementary volume *flavor[g]* defined by the flavor basis. This rule is valid in any space dimension. (see http://mathworld.wolfram.com/permutationTensor.html).

- *g(flavor→red) ≡ g = det[g_{ij}]* is the elementary volume determined by the red basis vectors. For convenience it is written as a tensor of order zero, but it is not invariant.

- For the standard orthonormal basis, *g(flavor→Identity) ≡ g = det[g_{ij}]* we can also define as the "black" basis using `DeclareIndexFlavor[{black,Identity}]`.

- See also: `PermutationSymbolRule`, `LeviCivitaOrder`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]
```

```
In[6]:= DefineTensorShortcuts[{e, 3}]
```

```
In[7]:= eddd[red@i, red@j, red@k] // LeviCivitaSimplify[e, ϵ][g, red]
        Tensor[T, red /@ {i, j, k}, {Void, Void, Void}] % // EinsteinSum[];
        % /. PermutationSymbolRule[ϵ] // Factor
```

Out[7]= $(g)^{\frac{1}{2}}\,\epsilon_{i\,j\,k}$

Out[9]= $(g)^{\frac{1}{2}}\,(T^{1\,2\,3} - T^{1\,3\,2} - T^{2\,1\,3} + T^{2\,3\,1} + T^{3\,1\,2} - T^{3\,2\,1})$

Restore the settings.

```
In[10]:= ClearTensorShortcuts[{e, 3}]
```

```
In[11]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

LowerIndexD

- `LowerIndexD[i, j][tensor]` will lower the upper index i or contravariant index i in tensor and rename it j.

- `LowerIndexD[i, j][expr]` will lower the index and contravariant index in all Tensors in expr.

- This routine is primarily used in programming other routines and in controlled circumstances.
- The indices i and j should be raw index Symbols.
- See also: `RaiseIndexD`, `ReplaceIndex`, `ParseTermIndices`, `MetricSimplifyD`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the old settings.

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;

In[5]:= DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]
        DefineTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]

In[7]:= Sdu[i, j]
        % // LowerIndexD[j, j]
        ContravariantD[Su[i], j]
        % // LowerIndexD[j, k]
```

Out[7]= $S_i{}^{;j}$

Out[8]= $S_{i\,j}$

Out[9]= $S^{i\,;j}$

Out[10]= $S^{i}{}_{;k}$

So that using flavored index has no influence, in particular it cannot be used together with a basis change :

```
In[11]:= res = ContravariantD[Su[i], j] // ToFlavor[red]
          res // LowerIndexD[red@j, red@k]
          res // LowerIndexD[red@j, blue@k]
```

Out[11]= $S^{i\,;j}$

Out[12]= $S^{i}{}_{;k}$

Out[13]= $S^{i}{}_{;k}$

When used on a specific Tensor an error is generated if the lower index does not exist.

```
In[14]:= ContravariantD[Su[i], j]
          % // LowerIndexD[k, j]
```

Out[14]= $S^{i\,;j}$

LowerIndex::noindex : k is not an upper index in $S^{i\,\ll 55 \gg\,(j)}$

Out[15]= $Aborted

When applied to an expression whose Head is not Tensor, the routine ignores Tensors that do not have the index.

```
In[16]:= ContravariantD[Sd[i], j] ContravariantD[Td[a], b]
          % // LowerIndexD[b, c]
```

Out[16]= $S_i{}^{;j}\,T_a{}^{;b}$

Out[17]= $T_{a\,;c}\,S_i{}^{;j}$

Restore the initial settings...

```
In[18]:= ClearTensorShortcuts[{{S, T}, 2}]

In[19]:= ClearIndexFlavor /@ IndexFlavors;
          DeclareIndexFlavor /@ oldflavors;
          Clear[oldflavors]
```

MetricRuleD

■ `MetricRuleD[x,g][expr]` rewrites covariant derivatives of basis coordinates x in expr, into the (mixed) metric tensor g, and simplify its value to 1 if the derivation is on the same coordinate and to 0 otherwise. The result is left non evaluated if an index of g is literal.

■ See also: `BasisChange`, `SetMetricValues`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]

In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}, {blue, Blue}]
        DeclareBaseIndices[{1, 2, 3}];

In[7]:= DefineTensorShortcuts[{{x}, 1}, {{g}, 2}];

In[8]:= CovariantD[xu[red@2], red@2]
        % // MetricRuleD[x, g]

Out[8]= x^2_;2

Out[9]= 1

In[10]:= CovariantD[xu[red@1], red@3]
         % // MetricRuleD[x, g]

Out[10]= x^1_;3

Out[11]= 0

In[12]:= CovariantD[xu[red@2], red@i]
         % // MetricRuleD[x, g]

Out[12]= x^2_;i

Out[13]= x^2_;i

In[14]:= CovariantD[xu[red@2], blue@3]
         % // MetricRuleD[x, g]

Out[14]= x^2_;3

Out[15]= x^2_;3

In[16]:= ClearTensorShortcuts[{{x}, 1}, {{g}, 2}];

In[17]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

MetricSimplifyD

■ `MetricSimplifyD[g][expression]` will raise or lower indices as well as covariant and contravariant indicies when products with the metric tensor g appear.

- g is the label of the metric tensor. The routine assumes that any second order tensor with the label g is a metric tensor. One could also use other labels to stand for the metric tensor, for example η or h.
- Simplification occurs only if the two indices of `g` have the same flavor.
- Up/down and down/up metric tensors will act as Kroneckers, performing index substitution. Up/down and down/up metric tensors that contain base indices will simplify to 0 or 1. Hence you can use MetricSimplify to evaluate Kroneckers.
- The decision to use `MetricSimplifyD` may depend upon the particular tensor forms that have stored values. You may often wish to use `MetricSimplifyD` on only portions of an expression using `MapLevelParts`.
- If both indices of a metric tensor are dummy indices in an expression, then the lowest sort order index is used as the active dummy index in the simplification.
- MetricSimplifyD will not commute with `PartialD` or `TotalD`.
- See also: `SetTensorValues`, `SetTensorValueRules`, `EvaluateDotProducts`, `MapLevelParts`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings and declare base indices and flavors...

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareBaseIndices[{1, 2, 3}]
        DeclareIndexFlavor /@ {{red, Red}, {black, Black}};

In[8]:= DefineTensorShortcuts[{{A, x, y}, 1}, {{g, h, η, δ, S, β}, 2}]

In[9]:= SetScalarSingleCovariantD[False]
```

Raising covariant indices...

```
In[10]:= guu[i, j] CovariantD[xu[k], i]
         % // MetricSimplifyD[g]
         % // FullForm // StandardForm

Out[10]= x^k_;i g^ij

Out[11]= x^k;j

Out[12]//StandardForm=
HoldContravariantD[Tensor[x, List[black[k]], List[Void]], black[j]]
```

Lowering contravariant indices....

```
In[13]:= gdd[i, m] gdd[j, n] ContravariantD[Suu[k, i], j]
         % // MetricSimplifyD[g]
         % // FullForm // StandardForm

Out[13]= S^ki;j g_im g_jn

Out[14]= S^k_m;n

Out[15]//StandardForm=
CovariantD[Tensor[S, List[black[k], Void], List[Void, black[m]]], black[n]]
```

Mixed up/down metric tensors act like Kroneckers.

```
In[16]:= gud[i, j] CovariantD[xu[k], i]
         % // MetricSimplifyD[g]

Out[16]= x^k_;i g^i_j

Out[17]= x^k_;j
```

MetricSimplifyD is mapped over equations, lists and sums.

```
In[18]:= guu[i, j] (CovariantD[xu[k], i] + CovariantD[yu[k], i])
         % // MetricSimplifyD[g]
         % == ContravariantD[xu[k] + yu[k], j]
```

Out[18]= $\left(x^k{}_{;i} + y^k{}_{;i}\right) g^{ij}$

Out[19]= $x^{k;j} + y^{k;j}$

Out[20]= True

Dummy indices...

```
In[21]:= guu[i, j] CovariantD[xu[k], i] CovariantD[yu[l], j]
         % // MetricSimplifyD[g]
```

Out[21]= $x^k{}_{;i}\, y^l{}_{;j}\, g^{ij}$

Out[22]= $y^l{}_{;j}\, x^{k;j}$

Scalar tensors... in this case covariant derivation is equivalent to a partial derivation

```
In[23]:= guu[i, j] CovariantD[Tensor[T], i]
         % // MetricSimplifyD[g]
         gdd[i, j] ContravariantD[Tensor[T], i]
         % // MetricSimplifyD[g]
```

Out[23]= $T_{;i}\, g^{ij}$

Out[24]= $T^{;j}$

Out[25]= $T^{;i}\, g_{ij}$

Out[26]= $T_{;j}$

```
In[27]:= guu[i, j] CovariantD[Tensor[T], i]
         % // MetricSimplifyD[g]
```

Out[27]= $T_{;i}\, g^{ij}$

Out[28]= $T^{;j}$

MetricSimplifyD commutes with Covariant and Contravariant derivatives. This can be done because the covariant derivative of the metric is zero.

```
In[29]:= gdd[a, b] CovariantD[Au[b], {c, d}]
         % // MetricSimplify[g]
```

Out[29]= $A^b{}_{;c\,d}\, g_{ab}$

Out[30]= $A_{a\,;c\,d}$

```
In[31]:= gdd[a, b] ContravariantD[Au[b], {c, d}]
         % // MetricSimplifyD[g]
```

Out[31]= $A^{b\,;c\,d}\, g_{ab}$

Out[32]= $A_a{}^{;c\,d}$

We can use `MapLevelParts` to do partial metric simplifications.

```
In[33]:= guu[i, m] guu[j, n] CovariantD[xu[k], i] CovariantD[yu[k], j]
         % // MapLevelParts[MetricSimplifyD[g], {{1, 3}}]
```

Out[33]= $x^k{}_{;i}\, y^k{}_{;j}\, g^{im}\, g^{jn}$

Out[34]= $y^k{}_{;j}\, x^{k\,;m}\, g^{jn}$

When an up/down or down/up metric tensor contains base indices it is evaluated to 0 or 1.

```
In[35]:= {gud[1, 1], gud[1, 2], guu[1, 1], gdd[1, 2], gud[red@1, red@1], gdu[red@1, red@2]};
         Thread[% → (% // MetricSimplifyD[g])]
```

Out[36]= $\{g^1{}_1 \to 1,\ g^1{}_2 \to 0,\ g^{11} \to g^{11},\ g_{12} \to g_{12},\ g^1{}_1 \to 1,\ g_1{}^2 \to 0\}$

```
In[37]:= δud[3, 3]
         % // MetricSimplifyD[δ]   (* is here equivalent to MetricSimplify[g] *)
```

Out[37]= $\delta^{3}{}_{3}$

Out[38]= 1

The base indicies don't have to be integers.

```
In[39]:= DeclareBaseIndices[{ρ, θ, ϕ}]
         {gud[ρ, ρ], gud[ρ, θ], guu[ρ, ρ], gdd[ρ, ϕ], gud[red@ρ, red@ρ], gdu[red@θ, red@ϕ]};
         Thread[% → (% // MetricSimplifyD[g])]
         DeclareBaseIndices[{1, 2, 3}]
```

Out[41]= $\{g^{\rho}{}_{\rho} \to 1, g^{\rho}{}_{\theta} \to 0, g^{\rho\rho} \to g^{\rho\rho}, g_{\rho\phi} \to g_{\rho\phi}, g^{\rho}{}_{\rho} \to 1, g_{\theta}{}^{\phi} \to 0\}$

The index flavors must match...

```
In[43]:= guu[i, j] (CovariantD[xu[red@k], red@i])
         % // MetricSimplifyD[g]
```

Out[43]= $x^{k}{}_{;i}\, g^{ij}$

Out[44]= $x^{k}{}_{;i}\, g^{ij}$

Both indices on the metric matrix must be the same flavor.

```
In[45]:= gdd[red@i, j] ContravariantD[xu[k], j]
         % // MetricSimplifyD[g]
```

Out[45]= $x^{k;j}\, g_{ij}$

Out[46]= $x^{k;j}\, g_{ij}$

... except in the case of basis change (we then prefer to use another notation, like β). KroneckerAbsorb can also be used in this case :

```
In[47]:= βud[red@i, j] xu[j]
         % → (% // MetricSimplifyD[β])
         %% → (%% // KroneckerAbsorb[β])
```

Out[47]= $x^{j}\, \beta^{i}{}_{j}$

Out[48]= $x^{j}\, \beta^{i}{}_{j} \to x^{i}$

Out[49]= $x^{j}\, \beta^{i}{}_{j} \to x^{i}$

MetricSimplify can use base indices as the replacement value.

```
In[50]:= guu[3, i] CovariantD[xu[k], i]
         % // MetricSimplifyD[g]
```

Out[50]= $x^{k}{}_{;i}\, g^{3i}$

Out[51]= $x^{k;3}$

MetricSimplify will not commute with partial or total differentiation

```
In[52]:= guu[i, j] PartialD[xu[m], j]
         % // MetricSimplifyD[g]
```

Out[52]= $x^{m}{}_{,j}\, g^{ij}$

Out[53]= $x^{m}{}_{,j}\, g^{ij}$

Restore the settings...

```
In[54]:= ClearTensorShortcuts[{{A, x, y}, 1}, {{g, h, η, S, β}, 2}]

In[55]:= DeclareBaseIndices[oldindices]
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

MetricTripleProduct

■ `MetricTripleProduct[basis,metric,LeviCivita,k][expr]` expresses any scalar triple products into combination of the components using Levi-Civita and metric tensors, k is a dummy index. Must not be confused with "ScalarTripleProduct", defined in the package Calculus`VectorAnalysis`"

■ See also: `CrossProductExpansion`, `LeviCivitaRule`, `LeviCivitaSimplify`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[6]:= DefineTensorShortcuts[{{u, v, w, e}, 1}]

In[7]:= u = uu[i] ed[i];
        v = vu[j] ed[j];
        w = wd[k] eu[k];

In[10]:= ((u × v).w // ToFlavor[red] // MetricTripleProduct[e, g, e, h])
         (% // UpDownSwap[red@k])
```

Warning : be sure that index 'h' does not appear in $(u^i \mathbf{e}_i)\times(v^j \mathbf{e}_j).(w_k \mathbf{e}^k)$

```
Out[10]= e_ij^k u^i v^j w_k

Out[11]= e_ijk u^i v^j w^k
```

Restore the settings.

```
In[12]:= ClearTensorShortcuts[{T, 2}, {T, 3}]

In[13]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

PartialDOrdering

■ `PartialDOrdering[expr]` orders the indices of partial derivations in expr, in a standard order.

■ See also: `SymmetricStandardOrder`, `CovariantDOrdering`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[7]:= DefineTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]

In[8]:= Td[red@m]
        PartialD[%, {red@k, red@j, red@i}]
        % // PartialDOrdering
```

```
Out[8]= T_m

Out[9]= T_m,kji

Out[10]= T_m,ijk
```

```
In[11]:= Tuu[red@n, red@m]
         PartialD[%, {red@k, red@j, red@i}]
         % // PartialDOrdering
```

Out[11]= $T^{n\,m}$

Out[12]= $T^{n\,m}{}_{,k\,j\,i}$

Out[13]= $T^{n\,m}{}_{,i\,j\,k}$

SymmetricStandardOrder will be used to order the other tensorial indicies :

```
In[14]:= PartialD[Tuu[red@n, red@m], {red@k, red@j, red@i}]
         % // SymmetricStandardOrder[T, {2}]
```

Out[14]= $T^{n\,m}{}_{,k\,j\,i}$

Out[15]= $T^{m\,n}{}_{,k\,j\,i}$

Restore settings.

```
In[16]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

```
In[17]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

PartialDSimplify

- PartialDSimplify[*expr*] extract outside the derivation the terms in the partial derivatives which do not depend on the coordinate of derivation.

- See also: PartialD, DifToPartialD.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}, {hat, OverHat}]
        labels = {x, δ, g, Γ};
```

To conserve a compact form, an expression like,

```
In[7]:= res = PartialD[{x, δ, g, Γ}][
          Tensor[Sin[Tensor[x, {red[2]}, {Void}]] * Tensor[v, {Void}, {hat[3]}] *
            Tensor[x, {red[1]}, {Void}]], Tensor[x, {red[1]}, {Void}]]
```

Out[7]= $\frac{\partial \sin(x^2)\, v_{\hat{3}}\, x^1}{\partial x^1}$

has been nested (with Tensor[expression]). Such an expression is not completely satisfactory, as the factor $\sin(x^2)$ does not depend on x^1 and can be factorized.

But UnNesting the expression expands it completely

```
In[8]:= PartialD[{x, δ, g, Γ}][Tensor[Sin[Tensor[x, {red[2]}, {Void}]] *
            Tensor[v, {Void}, {hat[3]}] * Tensor[x, {red[1]}, {Void}]],
          Tensor[x, {red[1]}, {Void}]] // UnnestTensor // Kronecker[δ]
```

Out[8]= $\sin(x^2)\left(v_{\hat{3}} + x^1 \frac{\partial v_{\hat{3}}}{\partial x^1}\right)$

If we want to obtain the following result,

```
In[9]:= Sin[Tensor[x, {red[2]}, {Void}]] *
         PartialD[{x, δ, g, Γ}][Tensor[Tensor[v, {Void}, {hat[3]}] Tensor[x, {red[1]}, {Void}]],
          Tensor[x, {red[1]}, {Void}]]
```

$$\text{Out[9]=}\quad \sin(x^2)\,\frac{\partial v_{\hat{3}}\,x^1}{\partial x^1}$$

then we can use PartialDSimplify :

```
In[10]:= res == (res // PartialDSimplify)
```

$$\text{Out[10]=}\quad \frac{\partial \sin(x^2)\,v_{\hat{3}}\,x^1}{\partial x^1} = \sin(x^2)\,\frac{\partial v_{\hat{3}}\,x^1}{\partial x^1}$$

```
In[11]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

PartialDToDif

■ `PartialDToDif[`*expr*`]` rewrites the partial derivatives in their expanded form present in expr, into the partial derivatives with respect to the index of the coordinates.

■ See also: `PartialD`, `DifToPartialD`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]
        labels = {x, δ, g, Γ};
```

```
In[7]:= DefineTensorShortcuts[{x, 1}, {T, 2}]
```

```
In[8]:= ((drv = PartialD[labels][Tensor[T, {i, j}, {Void, Void}], {xu[k], xu[h]}]) ==
           (drv // PartialDToDif))
```

$$\text{Out[8]=}\quad \frac{\partial^2\,T^{i\,j}}{\partial x^k\,\partial x^h} = T^{i\,j}{}_{,k\,h}$$

Restore the settings.

```
In[9]:= ClearTensorShortcuts[{x, 1}, {T, 2}]
```

```
In[10]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

RaiseIndexD

■ `RaiseIndexD[i, j][`*tensor*`]` will raise the lower index i in tensor, or the covariant index i, and rename it j.

■ `RaiseIndexD[i, j][`*expr*`]` will raise the index or covariant index in all Tensors in expr.

■ This routine is primarily used in programming other routines and in controlled circumstances.

■ The indices i and j should be raw index Symbols.

■ See also: `LowerIndexD`, `ReplaceIndex`, `ParseTermIndices`, `MetricSimplifyD`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the old settings.

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;

In[5]:= DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]
        DefineTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]

In[7]:= Sud[i, j]
        % // RaiseIndexD[j, j]

Out[7]= S^i_j

Out[8]= S^{i j}

In[9]:= CovariantD[Su[i], j]
        % // RaiseIndexD[j, j]

Out[9]= S^i_{;j}

Out[10]= S^{i ;j}
```

The raw index is used in the routine...

```
In[11]:= Sud[i, j] // ToFlavor[red]
         % // RaiseIndexD[j, k]
         CovariantD[Su[i], j] // ToFlavor[red]
         % // RaiseIndexD[j, k]

Out[11]= S^i_j

Out[12]= S^{i k}

Out[13]= S^i_{;j}

Out[14]= S^{i ;k}
```

So that using flavored index has no influence, in particular it cannot be used together with a basis change :

```
In[15]:= res = CovariantD[Sd[i], j] // ToFlavor[red]
         res // RaiseIndexD[red@j, red@k]
         res // RaiseIndexD[red@j, blue@k]

Out[15]= S_{i ;j}

Out[16]= S_i^{ ;k}

Out[17]= S_i^{ ;k}
```

When used on a specific Tensor an error is generated if the lower index does not exist.

```
In[18]:= CovariantD[Sd[i], j]
         % // RaiseIndexD[k, j]

Out[18]= S_{i ;j}

RaiseIndex::noindex : k is not a lower index in S_{i ≪56≫ (j)}

Out[19]= $Aborted
```

When applied to an expression whose Head is not Tensor, the routine ignores Tensors that do not have the index.

```
In[20]:= CovariantD[Sd[i], j] CovariantD[Td[a], b]
         % // RaiseIndexD[b, c]

Out[20]= S_{i ;j} T_{a ;b}

Out[21]= S_{i ;j} T_a^{ ;c}
```

Restore the initial settings...

```
In[22]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]

In[23]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

ReleaseHoldD

■ `ReleaseHoldD[expr]` expands the contravariant derivatives of previously unexpanded terms.

■ `ReleaseHoldD` is necessary to expand the contravariant derivatives, in their Hold Form. For covariant derivatives both `ReleaseHoldD` and

`ReleaseHold` can be used.

■ If the Hold Form has been created by mean of `Tensor[term]`, use `ReleaseHoldD` which internally applies an `UnnestTensor` operation.

■ See also: `HoldCovariantD`, `HoldContravariantD`, `HoldCovariantD2d`, `HoldContravariantD2d`, `HoldOp`,

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save the settings.

```
In[3]:= oldindices = BaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]

In[7]:= DefineTensorShortcuts[{{R, S, T}, 1}, {{R, S, T}, 2}]
```

ReleaseHoldD is used to expand the derivatives. In the case of covariant derivatives, **ReleaseHold** may also be used as the covariant derivatives are created in *Tensorial*.

```
In[8]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
        HoldCovariantD[%, red@k]
        % // ReleaseHoldD    (* or equivalently *)
        %% // ReleaseHold
```

$$\text{Out[8]=}\ \frac{2 a \pi S^i T^j}{\nu+1}$$

$$\text{Out[9]=}\ \left(\frac{2 a \pi S^i T^j}{\nu+1}\right)_{;k}$$

$$\text{Out[10]=}\ \frac{2 a \pi \left(T^j{}_{;k}\, S^i + S^i{}_{;k}\, T^j\right)}{\nu+1}$$

$$\text{Out[11]=}\ \frac{2 a \pi \left(T^j{}_{;k}\, S^i + S^i{}_{;k}\, T^j\right)}{\nu+1}$$

```
In[12]:= Tensor[2 π a / (1 + ν) Su[i] Tu[j]] // ToFlavor[red]
         CovariantD[%, red@k]
         % // ReleaseHoldD    (* but not *)
         %% // ReleaseHold
```

$$\text{Out[12]=}\ \frac{2 a \pi S^i T^j}{\nu+1}$$

$$\text{Out[13]=}\ \left(\frac{2 a \pi S^i T^j}{\nu+1}\right)_{;k}$$

$$\text{Out[14]=}\ \frac{2 a \pi \left(T^j{}_{;k}\, S^i + S^i{}_{;k}\, T^j\right)}{\nu+1}$$

$$\text{Out[15]=}\ \left(\frac{2 a \pi S^i T^j}{\nu+1}\right)_{;k}$$

```
In[16]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         HoldCovariantD2d[%, red@k]
         % // ReleaseHoldD    (* or equivalently *)
         %% // ReleaseHold
```

Out[16]= $\frac{2 a \pi S^i T^j}{\nu + 1}$

Out[17]= $\left(\frac{2 a \pi S^i T^j}{\nu + 1}\right)_{|k}$

Out[18]= $\frac{2 a \pi \left(T^j{}_{|k} S^i + S^i{}_{|k} T^j\right)}{\nu + 1}$

Out[19]= $\frac{2 a \pi \left(T^j{}_{|k} S^i + S^i{}_{|k} T^j\right)}{\nu + 1}$

But we must use **ReleaseHoldD** to expand contravariant derivatives

```
In[20]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         HoldContravariantD[%, red@k]
         % // ReleaseHoldD
```

Out[20]= $\frac{2 a \pi S^i T^j}{\nu + 1}$

Out[21]= $\left(\frac{2 a \pi S^i T^j}{\nu + 1}\right)^{:k}$

Out[22]= $\frac{2 a \pi \left(T^{j:k} S^i + S^{i:k} T^j\right)}{\nu + 1}$

```
In[23]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         HoldContravariantD2d[%, red@k]
         % // ReleaseHoldD
```

Out[23]= $\frac{2 a \pi S^i T^j}{\nu + 1}$

Out[24]= $\left(\frac{2 a \pi S^i T^j}{\nu + 1}\right)^{|k}$

Out[25]= $\frac{2 a \pi \left(T^{j|k} S^i + S^{i|k} T^j\right)}{\nu + 1}$

Restore settings.

```
In[26]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

```
In[27]:= DeclareBaseIndices[oldindices];
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors]
```

StandardDownIndices

- `StandardDownIndices[g][expr]` rewrites all the tensors in expr with only down indices, with the introduction of the metric tensor g. This allows to have a standard writing of expr. It is particularly convenient when the basis is orthonormal (g is the identity matrix, and in this case many authors do not even introduce upper indices.

- See also: `IndexFlavors`, `ToFlavor`, `Flavor`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]

In[6]:= DefineTensorShortcuts[{{T, e}, 1}, {{T, g}, 2}];

In[7]:= SetTensorValues[gdd[i, j], IdentityMatrix[3]]
        RedMetric = {{2, 1, 1}, {1, 5, 2}, {1, 2, 2}};
        SetTensorValueRules[gdd[red@i, red@j], RedMetric]

In[10]:= Tud[i, j]
         % // StandardDownIndices[g] // EinsteinSum[]
         (% // ArrayExpansion[black@i, black@j]) /. TensorValueRules[g]
```

$$\text{Out[10]=}\ T^{i}{}_{j}$$

$$\text{Out[11]=}\ g^{i1}\,T_{1j} + g^{i2}\,T_{2j} + g^{i3}\,T_{3j}$$

$$\text{Out[12]=}\ \begin{pmatrix} g^{11}T_{11}+g^{12}T_{21}+g^{13}T_{31} & g^{11}T_{12}+g^{12}T_{22}+g^{13}T_{32} & g^{11}T_{13}+g^{12}T_{23}+g^{13}T_{33} \\ g^{21}T_{11}+g^{22}T_{21}+g^{23}T_{31} & g^{21}T_{12}+g^{22}T_{22}+g^{23}T_{32} & g^{21}T_{13}+g^{22}T_{23}+g^{23}T_{33} \\ g^{31}T_{11}+g^{32}T_{21}+g^{33}T_{31} & g^{31}T_{12}+g^{32}T_{22}+g^{33}T_{32} & g^{31}T_{13}+g^{32}T_{23}+g^{33}T_{33} \end{pmatrix}$$

```
In[13]:= Tud[i, j] // ToFlavor[red]
         % // StandardDownIndices[g] // EinsteinSum[]
         (% // ArrayExpansion[red@i, red@j]) /. TensorValueRules[g]
```

$$\text{Out[13]=}\ T^{i}{}_{j}$$

$$\text{Out[14]=}\ g^{i1}\,T_{1j} + g^{i2}\,T_{2j} + g^{i3}\,T_{3j}$$

$$\text{Out[15]=}\ \begin{pmatrix} g^{11}T_{11}+g^{12}T_{21}+g^{13}T_{31} & g^{11}T_{12}+g^{12}T_{22}+g^{13}T_{32} & g^{11}T_{13}+g^{12}T_{23}+g^{13}T_{33} \\ g^{21}T_{11}+g^{22}T_{21}+g^{23}T_{31} & g^{21}T_{12}+g^{22}T_{22}+g^{23}T_{32} & g^{21}T_{13}+g^{22}T_{23}+g^{23}T_{33} \\ g^{31}T_{11}+g^{32}T_{21}+g^{33}T_{31} & g^{31}T_{12}+g^{32}T_{22}+g^{33}T_{32} & g^{31}T_{13}+g^{32}T_{23}+g^{33}T_{33} \end{pmatrix}$$

```
In[16]:= CovariantD[Tu[i], j]
         % // StandardDownIndices[g] // CovariantDSimplify[e, g, e] // EinsteinSum[]
         (% // ArrayExpansion[black@i, black@j]) /. TensorValueRules[g]
```

$$\text{Out[16]=}\ T^{i}{}_{;j}$$

$$\text{Out[17]=}\ T_{1\,;j}\,g^{i1} + T_{2\,;j}\,g^{i2} + T_{3\,;j}\,g^{i3}$$

$$\text{Out[18]=}\ \begin{pmatrix} T_{1;1}g^{11}+T_{2;1}g^{12}+T_{3;1}g^{13} & T_{1;2}g^{11}+T_{2;2}g^{12}+T_{3;2}g^{13} & T_{1;3}g^{11}+T_{2;3}g^{12}+T_{3;3}g^{13} \\ T_{1;1}g^{21}+T_{2;1}g^{22}+T_{3;1}g^{23} & T_{1;2}g^{21}+T_{2;2}g^{22}+T_{3;2}g^{23} & T_{1;3}g^{21}+T_{2;3}g^{22}+T_{3;3}g^{23} \\ T_{1;1}g^{31}+T_{2;1}g^{32}+T_{3;1}g^{33} & T_{1;2}g^{31}+T_{2;2}g^{32}+T_{3;2}g^{33} & T_{1;3}g^{31}+T_{2;3}g^{32}+T_{3;3}g^{33} \end{pmatrix}$$

```
In[19]:= ClearTensorShortcuts[{{T, e}, 1}, {T, 2}, {T, 3}]

In[20]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

SymmetricStandardOrder

- `SymmetricStandardOrder[x][expr]` allows to order vertical couples {flavor@i,Void}, ... of indices, irrespective of their flavor, in a symmetric tensor x present in *expr*. Upper indices are ordered first in alphabetic order on the left, then lower indices. It has some common behaviour with SymmetrizeSlots to which it is complementary.

`SymmetricStandardOrder[x, {ind}][expr]` orders the first 'ind' symmetrical indices.

`SymmetricStandardOrder[x, {ind1, ind2}][expr]` orders the symmetrical indices between columns 'ind1' and 'ind2'.

- See also: `SymmetrizeSlots`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}]

In[6]:= DefineTensorShortcuts[{T, 1}, {T, 2}, {T, 5}]

In[7]:= {Tddduu[red@k, h, blue@l, j, red@i], Tudduu[red@k, h, blue@l, j, red@i],
         Tuuuuu[red@k, h, blue@l, j, red@i], Tddddd[red@k, h, blue@l, j, red@i],
         Tuuuud[red@k, h, blue@l, j, red@i], Tudddd[red@k, h, blue@l, j, red@i]}
        % // SymmetricStandardOrder[T]
```

Out[7]= $\left\{T_{k\,h\,l}{}^{j\,i},\ T^{k}{}_{h\,l}{}^{j\,i},\ T^{k\,h\,l\,j\,i},\ T_{k\,h\,l\,j\,i},\ T^{k\,h\,l\,j}{}_{i},\ T^{k}{}_{h\,l\,j\,i}\right\}$

Out[8]= $\left\{T^{i\,j}{}_{h\,k\,l},\ T^{i\,j\,k}{}_{h\,l},\ T^{h\,i\,j\,k\,l},\ T_{h\,i\,j\,k\,l},\ T^{h\,j\,k\,l}{}_{i},\ T^{k}{}_{h\,i\,j\,l}\right\}$

```
In[9]:= ContravariantD[CovariantD[Tu[j], {k, j}], {i, k}]
        % // SymmetricStandardOrder[T]
```

Out[9]= $T^{j}{}_{;k\,j}^{\;;i\,k}$

Out[10]= $\left(T^{j\;;i\,k}\right)_{;j\,k}$

In the case of covariant and contravariant derivatives (and when the Gaussian curvature of the space is zero) :

```
In[11]:= term = ContravariantD[CovariantD[Tuu[red@b, red@a], {red@l, red@k}], {red@j, red@i}]
         term // SymmetricStandardOrder[T, {4, 5}]
         term // SymmetricStandardOrder[T, {3, 6}]
         term // SymmetricStandardOrder[T, {3, 4}]
         term // SymmetricStandardOrder[T, {1, 2}]
         % == term // SymmetricStandardOrder[T, {2}]
```

Out[11]= $T^{b\,a}{}_{;l\,k}^{\;;j\,i}$

Out[12]= $\left(T^{b\,a}{}_{;l}^{\;;j}\right)_{;k}^{\;;i}$

Out[13]= $\left(T^{b\,a\;;i\,j}\right)_{;k\,l}$

Out[14]= $T^{b\,a}{}_{;k\,l}^{\;;j\,i}$

Out[15]= $T^{a\,b}{}_{;l\,k}^{\;;j\,i}$

Out[16]= True

We cannot order mictures of tensor and derivative indices, they are ordered separately :

```
In[17]:= ContravariantD[CovariantD[Tuu[red@c, red@b], {red@k, red@a}], {red@j, red@i}]
         % // SymmetricStandardOrder[T, {4}]
         %% // SymmetricStandardOrder[T, {5}]
         %%% // SymmetricStandardOrder[T, {6}]
```

Out[17]= $T^{c\,b}{}_{;k\,a}^{\;;j\,i}$

Out[18]= $T^{b\,c}{}_{;a\,k}^{\;;j\,i}$

Out[19]= $\left(T^{b\,c\;;j}\right)_{;a\,k}^{\;;i}$

Out[20]= $\left(T^{b\,c\;;i\,j}\right)_{;a\,k}$

```
In[21]:= ClearTensorShortcuts[{T, 1}, {T, 2}, {T, 5}]

In[22]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

TCurl

■ TCurl[e,g,flavor/@{i,j,k},e][Vector] writes the curl of a vector field, in the basis e with a defined flavor, the metric tensor g, the dummy indices {i, j, k}, and the Levi-Civita symbol e.

■ See also: TGrad, TDiv, TLaplacian, LeviCivitaRule.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}, {blue, Blue}]

In[6]:= DefineTensorShortcuts[{{u, v, e}, 1}]

In[7]:= u = ud[red@i] eu[red@i];
        v = vu[red@i] ed[red@i];

In[9]:= TCurl[e, g, red /@ {i, j, k}, e][u]

Out[9]= u_{j;i} e^{ijk} e_k

In[10]:= TCurl[e, g, red /@ {i, j, k}, e][a u + b v]

Out[10]= (a u_{j;i} + b v_{j;i}) e^{ijk} e_k
```

If the Curl is taken in a different basis (u does not depend on the basis)

```
In[11]:= TCurl[e, g, blue /@ {p, q, r}, e][u]

Out[11]= u_{q;p} e^{pqr} e_r
```

Compare with **Curl** from "VectorAnalysis":

```
In[12]:= << Calculus`VectorAnalysis`
         Curl[{fx[x, y, z], fy[x, y, z], fz[x, y, z]}, Cartesian[x, y, z]]

Out[13]= {fz^(0,1,0)(x, y, z) - fy^(0,0,1)(x, y, z), fx^(0,0,1)(x, y, z) - fz^(1,0,0)(x, y, z), fy^(1,0,0)(x, y, z) - fx^(0,1,0)(x, y, z)}
```

Restore the settings.

```
In[14]:= ClearTensorShortcuts[{{u, v, e}, 1}]

In[15]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

TDiv

■ TDiv[*e,g,flavor@i*][*Vector*] writes the divergence of a vector field, in the basis e with a defined flavor, the metric tensor g, and the dummy index i.

■ See also: TGrad, TCurl, TLaplacian.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]

In[6]:= DefineTensorShortcuts[{{u, v, e}, 1}]

In[7]:= u = ud[red@i] eu[red@i];
        v = vu[red@i] ed[red@i];
        TDiv[e, g, red@k][u]

Out[9]= u^k_{;k}
```

```
In[10]:= TDiv[e, g, red@j][a u + b v]
```

$$\text{Out[10]}= a\, u^{j}{}_{;j} + b\, v^{j}{}_{;j}$$

Compare with **Div** from "VectorAnalysis" :

```
In[11]:= << Calculus`VectorAnalysis`
         Div[{fx[x, y, z], fy[x, y, z], fz[x, y, z]}, Cartesian[x, y, z]]
```

$$\text{Out[12]}= \text{fz}^{(0,0,1)}(x, y, z) + \text{fy}^{(0,1,0)}(x, y, z) + \text{fx}^{(1,0,0)}(x, y, z)$$

Restore the settings.

```
In[13]:= ClearTensorShortcuts[{{u, v, e}, 1}]

In[14]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

TensorComponentsD

■ `TensorComponentsD[g][expr]` simplifies in expr the derivation of a tensor component by another component of the same tensor (the components must of course be independent), g is the metric tensor.

■ See also: `PartialD`, `SetMetricValues`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[6]:= DefineTensorShortcuts[{T, 2}, {T, 3}]

In[7]:= a + b PartialD[Tudd[i, j, k], Tuu[p, q]]
        % // TensorComponentsD[g]
```

$$\text{Out[7]}= a + b\, T^{i}{}_{jk,T^{pq}}$$

$$\text{Out[8]}= a$$

```
In[9]:= a + b PartialD[Tudd[i, j, k], Tddd[p, q, r]]
        % // TensorComponentsD[g]
```

$$\text{Out[9]}= a + b\, T^{i}{}_{jk,T_{pqr}}$$

$$\text{Out[10]}= a + b\, g^{ip}\, g^{q}{}_{j}\, g^{r}{}_{k}$$

```
In[11]:= ClearTensorShortcuts[{T, 2}, {T, 3}]

In[12]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

TGrad

■ `TGrad[e,flavor@i][f]` writes the gradient of a scalar tensor f, in the basis e with a defined flavor, and the dummy index i.

■ See also: `TDiv`, `TCurl`, `TLaplacian`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]
```

TGrad acts on tensors,

```
In[6]:= Tf := Tensor[f]

In[7]:= TGrad[e, red@k][Tf]
```

Out[7]= $f_{,k}\, \mathbf{e}^{k}$

It may frequently be important to conserve the covariant notation in the case of covariant derivations of scalar tensors (even if the two are identical in this case). For that purpose we use **SetScalarSingleCovariantD[False]** the option **True** being the default option :

```
In[8]:= SetScalarSingleCovariantD[False]
        TGrad[e, red@k][Tf]
        SetScalarSingleCovariantD[True]
        TGrad[e, red@k][Tf]
        % == %%%
```

Out[9]= $f_{;k}\, \mathbf{e}^{k}$

Out[11]= $f_{,k}\, \mathbf{e}^{k}$

Out[12]= True

Compare with **Grad** from "VectorAnalysis" :

```
In[13]:= << Calculus`VectorAnalysis`
         Grad[a f[x, y, z] + b g[x, y, z], Cartesian[x, y, z]]
```

Out[14]= $\{a\, f^{(1,0,0)}(x, y, z) + b\, g^{(1,0,0)}(x, y, z),\ a\, f^{(0,1,0)}(x, y, z) + b\, g^{(0,1,0)}(x, y, z),\ a\, f^{(0,0,1)}(x, y, z) + b\, g^{(0,0,1)}(x, y, z)\}$

```
In[15]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

TLaplacian

- `TLaplacian[e,g,flavor@i,flavor@j][f]` calculates the laplacian of a scalar tensor f, in the basis e with a defined flavor, the metric tensor g, and the dummy indices i and j.
- `TLaplacian[e,g,flavor@i][f]` calculates the laplacian of a scalar tensor f, in the basis e with a defined flavor, the metric tensor g, and the dummy indices i and j.

- See also: `TGrad`, `TDiv`, `TCurl`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor...

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]
        DefineTensorShortcuts[{T, 1}, {T, 2}]
```

TLaplacian acts on tensors,

```
In[7]:= Tf := Tensor[f]
        TLaplacian[e, g, red@j, red@i][Tf]
        TLaplacian[e, g, red@i][Tf]
```

Out[8]= $f_{;i\,j}\, g^{i\,j}$

Out[9]= $(f^{i})_{;i}$

```
In[10]:= TLaplacian[e, g, red@i][Tu[red@j]]
```

Out[10]= $\left(T^{j;i}\right)_{;j}$

The standard Laplacian expression would give :

```
In[11]:= << Calculus`VectorAnalysis`
         Laplacian[a f[x, y, z] + b g[x, y, z], Cartesian[x, y, z]]
```

Out[12]= $a\, f^{(0,0,2)}(x, y, z) + b\, g^{(0,0,2)}(x, y, z) + a\, f^{(0,2,0)}(x, y, z) + b\, g^{(0,2,0)}(x, y, z) + a\, f^{(2,0,0)}(x, y, z) + b\, g^{(2,0,0)}(x, y, z)$

Restore the settings.

```
In[13]:= ClearTensorShortcuts[{T, 1}, {T, 2}]
```

```
In[14]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

ToFlavorD

■ `ToFlavorD[`*newflavor, oldflavor : black*`][expr]` will change all indices (tensors and derivations) in expr in the old flavor to the new flavor.

■ The second argument is optional. *black* corresponds to default black indices. (this is a difference with Tensorial 4, where Identity is the default flavor).

■ Base indices will be converted if they are integers, but otherwise will be left unchanged.

■ As an alternative, flavors can be directly entered in a tensor. For example `CovariantD[Sud[red@i, red@j], red@k]` but `ToFlavorD` checks that newflavor is a current flavor and issues an error message if not.

■ For a transformation matrix, which connects two coordinate systems, you will have to directly enter the flavors.

■ See also: `IndexFlavors`, `DeclareIndexFlavor`, `ClearIndexFlavor`, `IndexFlavorQ`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

The following saves the current state.

```
In[3]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
```

This declares new base indices, flavors and some tensor shortcuts...

```
In[6]:= DeclareIndexFlavor[{black, Black}, {red, Red}, {blue, Blue}, {rocket, SuperStar}];
        DeclareBaseIndices[{1, 2, 3}, {red, {x, y, z}}]
        DefineTensorShortcuts[{x, 1}, {{R, S, T}, 1}, {S, 2}]
```

```
In[9]:= HoldCovariantD[b^δ_γ u_δ, red[α]] a^γ_β
        ToFlavorD[rocket, red]@%
```

Out[9]= $\left(b^{\delta}{}_{\gamma}\, u_{\delta}\right)_{;\alpha}\, a^{\gamma}{}_{\beta}$

Out[10]= $\left(b^{\delta^*}{}_{\gamma^*}\, u_{\delta^*}\right)_{;\alpha^*}\, a^{\gamma^*}{}_{\beta^*}$

```
In[11]:= HoldCovariantD[b^δ_γ u_δ, red[α]]
         ToFlavorD[rocket, red]@%
```

Out[11]= $\left(b^{\delta}{}_{\gamma}\, u_{\delta}\right)_{;\alpha}$

Out[12]= $\left(b^{\delta^*}{}_{\gamma^*}\, u_{\delta^*}\right)_{;\alpha^*}$

```
In[13]:= CovariantD[b^δ_γ u_δ, red[α]]
         ToFlavorD[rocket, red]@%
```

Out[13]= $u_{\delta;\alpha}\, b^{\delta}{}_{\gamma} + b^{\delta}{}_{\gamma;\alpha}\, u_{\delta}$

Out[14]= $u_{\delta^*;\alpha^*}\, b^{\delta^*}{}_{\gamma^*} + b^{\delta^*}{}_{\gamma^*;\alpha^*}\, u_{\delta^*}$

```
In[15]:= HoldCovariantD2d[b^δ_γ u_δ, red[α]] a^γ_β + b_αγ a^γ_β
         ToFlavorD[rocket, red]@%
```

$$\text{Out[15]}= \left((b^{\delta}{}_{\gamma}\, u_{\delta})_{\mid \alpha}\, a^{\gamma}{}_{\beta} + a^{\gamma}{}_{\beta}\, b_{\alpha\gamma}\right)\left(u_{\delta^*;\alpha^*}\, b^{\delta^*}{}_{\gamma^*} + b^{\delta^*}{}_{\gamma^*;\alpha^*}\, u_{\delta^*}\right)$$

```
In[16]:= HoldContravariantD[b^δ_γ u_δ, red[α]] a^γ_β
         ToFlavorD[rocket, red]@%
```

$$\text{Out[16]}= (b^{\delta}{}_{\gamma}\, u_{\delta})^{;\alpha}\, a^{\gamma}{}_{\beta}$$

$$\text{Out[17]}= (b^{\delta^*}{}_{\gamma^*}\, u_{\delta^*})^{;\alpha^*}\, a^{\gamma^*}{}_{\beta^*}$$

```
In[18]:= HoldContravariantD[b^δ_γ u_δ, red[α]]
         ToFlavorD[rocket, red]@%%
```

$$\text{Out[18]}= (b^{\delta}{}_{\gamma}\, u_{\delta})^{;\alpha}$$

$$\text{Out[19]}= (b^{\delta^*}{}_{\gamma^*}\, u_{\delta^*})^{;(\alpha^*)^*}\, a^{\gamma^*}{}_{\beta^*}$$

```
In[20]:= HoldContravariantD2d[b^δ_γ u_δ, red[α]] a^γ_β + b_αγ a^γ_β
         ToFlavorD[rocket, red]@%
```

$$\text{Out[20]}= (b^{\delta^*}{}_{\gamma^*}\, u_{\delta^*})^{;\alpha^*}\, a^{\gamma^*}{}_{\beta^*}\, \left((b^{\delta}{}_{\gamma}\, u_{\delta})_{\mid \alpha}\, a^{\gamma}{}_{\beta} + a^{\gamma}{}_{\beta}\, b_{\alpha\gamma}\right)$$

```
In[21]:= ContravariantD2d[b^δ_γ u_δ, red[α]] a^γ_β + b_αγ a^γ_β
         ToFlavorD[rocket, red]@%
```

$$\text{Out[21]}= a^{\gamma}{}_{\beta}\, b_{\alpha\gamma} + a^{\gamma}{}_{\beta}\left(u_{\delta}{}^{\mid \alpha}\, b^{\delta}{}_{\gamma} + b^{\delta}{}_{\gamma}{}^{\mid \alpha}\, u_{\delta}\right)$$

$$\text{Out[22]}= a^{\gamma^*}{}_{\beta^*}\, b_{\alpha^*\gamma^*} + a^{\gamma^*}{}_{\beta^*}\left(u_{\delta^*}{}^{\mid \alpha^*}\, b^{\delta^*}{}_{\gamma^*} + b^{\delta^*}{}_{\gamma^*}{}^{\mid \alpha^*}\, u_{\delta^*}\right)$$

The second argument is optional. *black* corresponds to default black indices.

```
In[23]:= HoldContravariantD[Ru[i] Su[i] Tu[j], {m, n, q}]
         % // ToFlavorD[red]
```

$$\text{Out[23]}= \left(R^{i}\, S^{i}\, T^{j}\right)^{;mnq}$$

$$\text{Out[24]}= \left(R^{i}\, S^{i}\, T^{j}\right)^{;mnq}$$

```
In[25]:= HoldContravariantD[Ru[i] Su[i] Tu[j], {m, n, q}]
         % // ToFlavorD[red, black]
         % // ReleaseHoldD
         % // ToFlavorD[blue, red]
```

$$\text{Out[25]}= \left(R^{i}\, S^{i}\, T^{j}\right)^{;mnq}$$

$$\text{Out[26]}= \left(R^{i}\, S^{i}\, T^{j}\right)^{;mnq}$$

$$\begin{aligned}\text{Out[27]}= {} & R^{i;q}\, S^{i;n}\, T^{j;m} + R^{i;n}\, S^{i;q}\, T^{j;m} + R^{i;q}\, S^{i;m}\, T^{j;n} + R^{i;m}\, S^{i;q}\, T^{j;n} + R^{i;n}\, S^{i;m}\, T^{j;q} + R^{i;m}\, S^{i;n}\, T^{j;q} + \\ & \left(S^{i;n}\, T^{j;mq} + S^{i;nq}\, T^{j;m}\right) R^{i} + \left(S^{i;m}\, T^{j;nq} + S^{i;mq}\, T^{j;n}\right) R^{i} + R^{i;q}\, T^{j;mn}\, S^{i} + \left(R^{i;n}\, T^{j;mq} + R^{i;nq}\, T^{j;m}\right) S^{i} + \\ & \left(R^{i;m}\, T^{j;nq} + R^{i;mq}\, T^{j;n}\right) S^{i} + R^{i}\left(S^{i;q}\, T^{j;mn} + T^{j;mnq}\, S^{i}\right) + R^{i;q}\, S^{i;mn}\, T^{j} + \left(R^{i;n}\, S^{i;mq} + R^{i;nq}\, S^{i;m}\right) T^{j} + \\ & \left(R^{i;m}\, S^{i;nq} + R^{i;mq}\, S^{i;n}\right) T^{j} + R^{i;mn}\, S^{i;q}\, T^{j} + S^{i}\left(R^{i;mn}\, T^{j;q} + R^{i;mnq}\, T^{j}\right) + R^{i}\left(S^{i;mn}\, T^{j;q} + S^{i;mnq}\, T^{j}\right)\end{aligned}$$

$$\begin{aligned}\text{Out[28]}= {} & R^{i;q}\, S^{i;n}\, T^{j;m} + R^{i;n}\, S^{i;q}\, T^{j;m} + R^{i;q}\, S^{i;m}\, T^{j;n} + R^{i;m}\, S^{i;q}\, T^{j;n} + R^{i;n}\, S^{i;m}\, T^{j;q} + \\ & R^{i;m}\, S^{i;n}\, T^{j;q} + \left(S^{i;nq}\, T^{j;m} + S^{i;n}\, T^{j;mq}\right) R^{i} + \left(S^{i;mq}\, T^{j;n} + S^{i;m}\, T^{j;nq}\right) R^{i} + \\ & R^{i;q}\, T^{j;mn}\, S^{i} + \left(R^{i;nq}\, T^{j;m} + R^{i;n}\, T^{j;mq}\right) S^{i} + \left(R^{i;mq}\, T^{j;n} + R^{i;m}\, T^{j;nq}\right) S^{i} + \\ & R^{i}\left(S^{i;q}\, T^{j;mn} + T^{j;mnq}\, S^{i}\right) + R^{i;mn}\, S^{i;q}\, T^{j} + R^{i;q}\, S^{i;mn}\, T^{j} + \left(R^{i;nq}\, S^{i;m} + R^{i;n}\, S^{i;mq}\right) T^{j} + \\ & \left(R^{i;mq}\, S^{i;n} + R^{i;m}\, S^{i;nq}\right) T^{j} + S^{i}\left(R^{i;mn}\, T^{j;q} + R^{i;mnq}\, T^{j}\right) + R^{i}\left(S^{i;mn}\, T^{j;q} + S^{i;mnq}\, T^{j}\right)\end{aligned}$$

```
In[29]:= ContravariantD[Tensor[R, {red[i]}, {Void}] * Tensor[S, {red[i]}, {Void}] *
           Tensor[T, {red[j]}, {Void}], {red[m], red[n], red[q]}]
```

$$\begin{aligned}\text{Out[29]}= {} & R^{i;q}\, S^{i;n}\, T^{j;m} + R^{i;n}\, S^{i;q}\, T^{j;m} + R^{i;q}\, S^{i;m}\, T^{j;n} + R^{i;m}\, S^{i;q}\, T^{j;n} + R^{i;n}\, S^{i;m}\, T^{j;q} + R^{i;m}\, S^{i;n}\, T^{j;q} + \\ & \left(S^{i;n}\, T^{j;mq} + S^{i;nq}\, T^{j;m}\right) R^{i} + \left(S^{i;m}\, T^{j;nq} + S^{i;mq}\, T^{j;n}\right) R^{i} + R^{i;q}\, T^{j;mn}\, S^{i} + \left(R^{i;n}\, T^{j;mq} + R^{i;nq}\, T^{j;m}\right) S^{i} + \\ & \left(R^{i;m}\, T^{j;nq} + R^{i;mq}\, T^{j;n}\right) S^{i} + R^{i}\left(S^{i;q}\, T^{j;mn} + T^{j;mnq}\, S^{i}\right) + R^{i;q}\, S^{i;mn}\, T^{j} + \left(R^{i;n}\, S^{i;mq} + R^{i;nq}\, S^{i;m}\right) T^{j} + \\ & \left(R^{i;m}\, S^{i;nq} + R^{i;mq}\, S^{i;n}\right) T^{j} + R^{i;mn}\, S^{i;q}\, T^{j} + S^{i}\left(R^{i;mn}\, T^{j;q} + R^{i;mnq}\, T^{j}\right) + R^{i}\left(S^{i;mn}\, T^{j;q} + S^{i;mnq}\, T^{j}\right)\end{aligned}$$

```
In[30]:= ClearTensorShortcuts[{x, 1}, {{R, S, T}, 1}, {S, 2}]
```

This resets to the original state.

```
In[31]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor[IndexFlavors];
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

UpDownSwapD

■ UpDownSwapD [{*i1, i2, i3, ...*}, {*j1, j2, j3, ...*}] [*term*] will swap the up and down positions of the dummy indices {i1,i2,i3,...} in a term consisting of simple Tensor products, including covariant and contravariant derivatives, and rename them {j1,j2,j3,...}. It generalizes UpDownSwap[{i1,i2,i3,...},{j1,j2,j3,...}][term] of *Tensorial* 4.0

■ The dummy index must contain the flavor.

■ UpDownSwapD generalizes UpDownSwap to both tensor index and Covariant/Contravariant index.

■ UpDownSwapD is automatically mapped onto arrays, equations and sums.

■ UpDownSwapD does not check if the indices are dummy indices, it only changes up indices into down indices and vice-versa.

■ See also: IndexChange, TensorSimplify, SimplifyTensorSum, MetricSimplifyD.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]
```

Save old settings and declare a flavor.

```
In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]

In[6]:= DefineTensorShortcuts[{{u, X, usec}, 1}, {{ϵ}, 3}]

In[7]:= usec = OverDot[u, 2];

In[8]:= expr = μ CovariantD[ContravariantD[ud[red@l], red@m], {red@m, red@k}]
           ϵuuu[red[k], red[l], red[h]] +
         CovariantD[Xd[red@l], red@k] ϵuuu[red[k], red[l], red[h]] -
         ρ CovariantD[usecd[red@l], red[k]] ϵuuu[red[k], red[l], red[h]]
```

$$\mu\,(u_l{}^{;m})_{;m\,k}\,\epsilon^{k\,l\,h} + X_{l\,;k}\,\epsilon^{k\,l\,h} - \rho\,\ddot{u}_{l\,;k}\,\epsilon^{k\,l\,h}$$

```
In[9]:= expr // UpDownSwapD[red@l] // UpDownSwapD[red@k]
```

$$\mu\,(u^{l\,;m})_{;m}{}^{;k}\,\epsilon_{k\,l}{}^{h} + X^{l\,;k}\,\epsilon_{k\,l}{}^{h} - \rho\,\ddot{u}^{l\,;k}\,\epsilon_{k\,l}{}^{h}$$

```
In[10]:= expr // UpDownSwapD[{red@k, red@l}]
```

$$\mu\,(u^{l\,;m})_{;m}{}^{;k}\,\epsilon_{k\,l}{}^{h} + X^{l\,;k}\,\epsilon_{k\,l}{}^{h} - \rho\,\ddot{u}^{l\,;k}\,\epsilon_{k\,l}{}^{h}$$

but (warning !) :

```
In[11]:= expr // UpDownSwapD[red@h]
```

$$\mu\,(u_l{}^{;m})_{;m\,k}\,\epsilon^{k\,l}{}_{h} + X_{l\,;k}\,\epsilon^{k\,l}{}_{h} - \rho\,\ddot{u}_{l\,;k}\,\epsilon^{k\,l}{}_{h}$$

Restore old settings...

```
In[12]:= ClearTensorShortcuts[{{x, c}, 1}, {{a, b}, 2}]

In[13]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

VolumeSquare

■ VolumeSquare [*flavor@g*] [*expr*] evaluates the determinant of the metric tensor g in the flavor basis.

■ The evaluation is possible when *TensorValueRules[g]* has been defined.

■ VolumeSquare[*flavor@g*][*expr*] represents the square of the volume defined by the basis vectors in the flavor basis.

■ See also: `LeviCivitaSimplify`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        Needs["TContinuumMechanics2`TContinuumMechanics`"]

In[3]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{black, Black}, {red, Red}]
        DeclareBaseIndices[{1, 2, 3}];

In[7]:= DefineTensorShortcuts[{g, 2}, {e, 3}];

In[8]:= SetTensorValues[gdd[i, j], IdentityMatrix[3]]
        RedBasisVectors = {{1, 1, 0}, {0, 1, 2}, {1, 0, 1}}; RedMetric = {{2, 1, 1}, {1, 5, 2}, {1, 2, 2}};
        SetTensorValueRules[gdd[red@i, red@j], RedMetric]
        SetTensorValueRules[guu[red@i, red@j], Inverse[RedMetric]]

In[12]:= Tensor[red[g]]
         % // VolumeSquare[red@g]

Out[12]= g

Out[13]= 9

In[14]:= eddd[red@i, red@j, red@k]
         % // LeviCivitaSimplify[e, ϵ][g, red]
         Tensor[T, red /@ {i, j, k}, {Void, Void, Void}] % // EinsteinSum[];
         % /. PermutationSymbolRule[ϵ] // Factor

         % // VolumeSquare[red@g]

Out[14]= e_{i j k}

Out[15]= (g)^{1/2} ϵ_{i j k}

Out[17]= (g)^{1/2} (T^{1 2 3} - T^{1 3 2} - T^{2 1 3} + T^{2 3 1} + T^{3 1 2} - T^{3 2 1})

Out[18]= 3 (T^{1 2 3} - T^{1 3 2} - T^{2 1 3} + T^{2 3 1} + T^{3 1 2} - T^{3 2 1})

In[19]:= ClearTensorShortcuts[{g, 2}, {e, 3}];
```

This resets to the old indices.

```
In[20]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```