UnnestTensor

■ `UnnestTensor[`*expr*`]` removes Tensor wrappers from Tensor expressions.

In derivations and exposition it may be desirable to show derivatives of expressions in an unexpanded form. Wrapping Tensor expressions in a Tensor head prevents evaluation and expansion of derivatives. UnnestTensor removes the wrappers and allows evaluation and expansion.

Singly nested Tensor expressions can be parsed by `ParseTermIndices` provided they are free of Plus expressions.

Partial derivatives of nested tensors can be expanded to coordinates using `ExpandPartialD` but the same cannot be done for covariant derivatives using `ExpandCovariantD`.

See also: `Tensor`.

Example

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

In[2]:= DefineTensorShortcuts[{u, 1}]
```

Sometimes in presenting a derivation we don't want all derivatives to be immediately expanded. We can prevent this by initially enclosing an expression in Tensor.

Verify that...

```
In[3]:= Implies[uu[ν] ud[ν] == 1, CovariantD[uu[ν], μ] ud[ν] == 0] // TraditionalForm
```

```
Out[3]//TraditionalForm=
```

$$u_\nu\, u^\nu = 1 \Rightarrow u^\nu{}_{;\mu}\, u_\nu = 0$$

```
In[4]:= Tensor[uu[ν] ud[ν]] == 1
        Print["Covariant derivative of both sides"]
        CovariantD[#, μ] & /@ %%
        Print["Expand derivative by using UnnestTensor"]
        %% // UnnestTensor
        Print["Do an up down swap of ν index in the first term and simplify"]
        MapAt[UpDownSwap[ν], %%, {1, 1}]
        # / 2 & /@ % // FrameBox // DisplayForm
```

$$\text{Out[4]=}\quad u_\nu\, u^\nu == 1$$

Covariant derivative of both sides

$$\text{Out[6]=}\quad (u_\nu\, u^\nu)_{;\mu} == 0$$

Expand derivative by using UnnestTensor

$$\text{Out[8]=}\quad u^\nu{}_{;\mu}\, u_\nu + u_{\nu;\mu}\, u^\nu == 0$$

Do an up down swap of ν index in the first term and simplify

$$\text{Out[10]=}\quad 2\, u_{\nu;\mu}\, u^\nu == 0$$

Out[11]//DisplayForm=

$$\boxed{u_{\nu;\mu}\, u^\nu == 0}$$

ParseTermIndices will work on nested tensors provided they contain no Plus expressions. Then operations such as EinsteinSum or EinsteinArray can be used.

```
In[12]:= Tensor[ud[a] uu[a]]
         PartialD[%, b]
         step1 = % // ExpandPartialD[{x, δ, g, Γ}]
         % // EinsteinSum[]
         % // EinsteinArray[]
         % // UnnestTensor
```

$$\text{Out[12]=}\quad u^a\, u_a$$

$$\text{Out[13]=}\quad (u^a\, u_a)_{,b}$$

$$\text{Out[14]=}\quad \frac{\partial u^a\, u_a}{\partial x^b}$$

$$\text{Out[15]=}\quad \frac{\partial u^1\, u_1}{\partial x^b} + \frac{\partial u^2\, u_2}{\partial x^b} + \frac{\partial u^3\, u_3}{\partial x^b}$$

$$\text{Out[16]=}\quad \left\{\frac{\partial u^1\, u_1}{\partial x^1} + \frac{\partial u^2\, u_2}{\partial x^1} + \frac{\partial u^3\, u_3}{\partial x^1},\ \frac{\partial u^1\, u_1}{\partial x^2} + \frac{\partial u^2\, u_2}{\partial x^2} + \frac{\partial u^3\, u_3}{\partial x^2},\ \frac{\partial u^1\, u_1}{\partial x^3} + \frac{\partial u^2\, u_2}{\partial x^3} + \frac{\partial u^3\, u_3}{\partial x^3}\right\}$$

$$\text{Out[17]=}\quad \left\{u_1 \frac{\partial u^1}{\partial x^1} + u_2 \frac{\partial u^2}{\partial x^1} + u_3 \frac{\partial u^3}{\partial x^1} + u^1 \frac{\partial u_1}{\partial x^1} + u^2 \frac{\partial u_2}{\partial x^1} + u^3 \frac{\partial u_3}{\partial x^1},\right.$$
$$u_1 \frac{\partial u^1}{\partial x^2} + u_2 \frac{\partial u^2}{\partial x^2} + u_3 \frac{\partial u^3}{\partial x^2} + u^1 \frac{\partial u_1}{\partial x^2} + u^2 \frac{\partial u_2}{\partial x^2} + u^3 \frac{\partial u_3}{\partial x^2},$$
$$\left. u_1 \frac{\partial u^1}{\partial x^3} + u_2 \frac{\partial u^2}{\partial x^3} + u_3 \frac{\partial u^3}{\partial x^3} + u^1 \frac{\partial u_1}{\partial x^3} + u^2 \frac{\partial u_2}{\partial x^3} + u^3 \frac{\partial u_3}{\partial x^3}\right\}$$

As above, partial derivatives of nested expressions can be expanded to coordinates but the same cannot be done for covariant derivatives because those routines must know the form of the tensor expression.

```
In[18]:= ClearTensorShortcuts[{u, 1}]
```

UpDownAdjust

- `UpDownAdjust[`*sum*`]` will attemp to simplify terms that differ only in the up/down configuration of their dummy indices.

`UpDownAdjust` can be used after `TensorSimplify` to attempt to further simplify an expression consisting of a sum of terms. It is not automatiacally used because it might cause many changes in term configuration that may be unwanted.

See also: `TensorSimplify`, `UpDownSwap`, `UpDownAdjust`.

Examples

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

In[2]:= DefineTensorShortcuts[{{x, p, λ}, 1}, {{R, T}, 2}, {R, 4}]
        labs99 = {x, δ, g, Γ};

In[4]:= Tud[a, b] Rud[b, a] + Tdu[a, b] Rdu[b, a] + 3
        % // UpDownAdjust
```

$$\text{Out[4]= } 3 + R^{b}{}_{a}\, T^{a}{}_{b} + R_{b}{}^{a}\, T_{a}{}^{b}$$

$$\text{Out[5]= } 3 + 2\, R^{b\,a}\, T_{a\,b}$$

```
In[6]:= pu[a] Tdd[a, b] + pd[a] Tud[a, b] + λd[b]
        % // UpDownAdjust
```

$$\text{Out[6]= } p_{a}\, T^{a}{}_{b} + p^{a}\, T_{a\,b} + \lambda_{b}$$

$$\text{Out[7]= } 2\, p^{a}\, T_{a\,b} + \lambda_{b}$$

```
In[8]:= λu[c] PartialD[labs99][Tuu[a, b], xu[c]] + λd[c] PartialD[labs99][Tuu[a, b], xd[c]]
        % // UpDownAdjust
```

$$\text{Out[8]= } \lambda^{c}\, \frac{\partial T^{a\,b}}{\partial x^{c}} + \lambda_{c}\, \frac{\partial T^{a\,b}}{\partial x_{c}}$$

$$\text{Out[9]= } 2\, \lambda^{c}\, \frac{\partial T^{a\,b}}{\partial x^{c}}$$

`TensorSimplify` leaves a number of terms in the following expression unconsolidated.

```
In[10]:= Tensor[ϕ] + pu[a] pd[a]
         CovariantD[%, {b, c}];
         % // ExpandCovariantD[labs99, {d, e}];
         test = % // TensorSimplify
```

$$\text{Out[10]= } \phi + p^{a}\, p_{a}$$

$$\text{Out[13]= } p^{d}\, p_{a}\, \Gamma^{a}{}_{c\,e}\, \Gamma^{e}{}_{b\,d} - p^{d}\, p_{e}\, \Gamma^{a}{}_{b\,d}\, \Gamma^{e}{}_{c\,a} + \frac{\partial^{2}\phi}{\partial x^{c}\, \partial x^{b}} - \Gamma^{e}{}_{c\,b}\, \frac{\partial \phi}{\partial x^{e}} + p_{a}\, \frac{\partial^{2} p^{a}}{\partial x^{c}\, \partial x^{b}} -$$
$$p_{a}\, \Gamma^{e}{}_{c\,b}\, \frac{\partial p^{a}}{\partial x^{e}} + p^{a}\, \frac{\partial^{2} p_{a}}{\partial x^{c}\, \partial x^{b}} + \frac{\partial p^{a}}{\partial x^{c}}\, \frac{\partial p_{a}}{\partial x^{b}} + \frac{\partial p^{a}}{\partial x^{b}}\, \frac{\partial p_{a}}{\partial x^{c}} - p^{a}\, \Gamma^{e}{}_{c\,b}\, \frac{\partial p_{a}}{\partial x^{e}}$$

`UpDownAdjust` consolidates most of the remaining terms but can't recognize the two first partial products. `UpDownSwap` completes the simplification.

```
In[14]:= test // UpDownAdjust
         MapAt[UpDownSwap[a], %, 7]
```

$$\text{Out[14]}= p^d\, p_a\, \Gamma^a{}_{ce}\, \Gamma^e{}_{bd} - p^d\, p_e\, \Gamma^a{}_{bd}\, \Gamma^e{}_{ca} + \frac{\partial^2 \phi}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb}\, \frac{\partial \phi}{\partial x^e} + 2\, p^a\, \frac{\partial^2 p_a}{\partial x^c\, \partial x^b} + \frac{\partial p^a}{\partial x^c}\, \frac{\partial p_a}{\partial x^b} + \frac{\partial p^a}{\partial x^b}\, \frac{\partial p_a}{\partial x^c} - 2\, p^a\, \Gamma^e{}_{cb}\, \frac{\partial p_a}{\partial x^e}$$

$$\text{Out[15]}= p^d\, p_a\, \Gamma^a{}_{ce}\, \Gamma^e{}_{bd} - p^d\, p_e\, \Gamma^a{}_{bd}\, \Gamma^e{}_{ca} + \frac{\partial^2 \phi}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb}\, \frac{\partial \phi}{\partial x^e} + 2\, p^a\, \frac{\partial^2 p_a}{\partial x^c\, \partial x^b} + 2\, \frac{\partial p^a}{\partial x^c}\, \frac{\partial p_a}{\partial x^b} - 2\, p^a\, \Gamma^e{}_{cb}\, \frac{\partial p_a}{\partial x^e}$$

```
In[16]:= ClearTensorShortcuts[{{x, p, λ}, 1}, {{R, T}, 2}, {R, 4}]

In[17]:= Clear[test, labs99]
```

UpDownSwap

- `UpDownSwap[`*dummy*`][`*term*`]` will swap the up and down positions of the dummy index in a term consisting of simple Tensor products.

The dummy index must contain the flavor.

`UpDownSwap` is automatically mapped onto arrays, equations and sums.

See also: `IndexChange`, `DummySimplify`, `SimplifyTensorSum`, `MetricSimplify`.

Examples

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

Save old settings and declare a flavor.

```
In[2]:= oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]

In[5]:= DefineTensorShortcuts[{{x, c}, 1}, {{a, b}, 2}]

In[6]:= 2 auu[i, j] bdd[j, k] cd[i]
        Print["Swap i index"]
        %% // UpDownSwap[i]
        Print["Swap j index"]
        %% // UpDownSwap[j]
        Print["Swapping k does nothing because it is not a dummy index"]
        %% // UpDownSwap[k]

Out[6]= 2 a^{i j} b_{j k} c_i

          Swap i index

Out[8]= 2 a_i^j b_{j k} c^i

          Swap j index

Out[10]= 2 a_{i j} b^j_k c^i

          Swapping k does nothing because it is not a dummy index

Out[12]= 2 a_{i j} b^j_k c^i
```

The flavor must be on the index.

```
In[13]:= 2 auu[i, j] bdd[j, k] cd[i] // ToFlavor[red]
         Print["Swap i index"]
         %% // UpDownSwap[red@i]
         Print["Swap j index"]
         %% // UpDownSwap[red@j]
         Print["Swapping i without the color has no effect"]
         %% // UpDownSwap[i]
```

Out[13]= $2\, a^{i\,j}\, b_{j\,k}\, c_{i}$

```
Swap i index
```

Out[15]= $2\, a_{i}{}^{j}\, b_{j\,k}\, c^{i}$

```
Swap j index
```

Out[17]= $2\, a_{i\,j}\, b^{j}{}_{k}\, c^{i}$

```
Swapping i without the color has no effect
```

Out[19]= $2\, a_{i\,j}\, b^{j}{}_{k}\, c^{i}$

UpDownSwap is mapped over lists, single equations and sums, but not other subexpressions.

```
In[20]:= {xu[i] cd[i] + bud[i, i] == aud[i, i], bud[i, j] xu[j] ≥ aud[i, j] cu[j]}
         Print["Swapping i works on first equation"]
         %% // UpDownSwap[i]
         Print["Swapping j works on second equation"]
         %% // UpDownSwap[j]
```

Out[20]= $\left\{b^{i}{}_{i} + c_{i}\, x^{i} == a^{i}{}_{i},\ b^{i}{}_{j}\, x^{j} \geq a^{i}{}_{j}\, c^{j}\right\}$

```
Swapping i works on first equation
```

Out[22]= $\left\{b_{i}{}^{i} + c^{i}\, x_{i} == a_{i}{}^{i},\ b^{i}{}_{j}\, x^{j} \geq a^{i}{}_{j}\, c^{j}\right\}$

```
Swapping j works on second equation
```

Out[24]= $\left\{b_{i}{}^{i} + c^{i}\, x_{i} == a_{i}{}^{i},\ b^{i\,j}\, x_{j} \geq a^{i\,j}\, c_{j}\right\}$

UpDownSwap will work on any term that ParseTermIndices can extract the indices.

```
In[25]:= {adu[i, j] xd[j], xu[j] PartialD[{x, δ, g, Γ}][cd[i], xu[j]],
          (k1 xu[j]) . (k2 cd[j]), xu[j] ⊗ cd[j]}
         % // UpDownSwap[j]
```

Out[25]= $\left\{a_{i}{}^{j}\, x_{j},\ x^{j}\, \frac{\partial c_{i}}{\partial x^{j}},\ \left(k1\, x^{j}\right) . \left(k2\, c_{j}\right),\ x^{j} \otimes c_{j}\right\}$

Out[26]= $\left\{a_{i\,j}\, x^{j},\ x_{j}\, \frac{\partial c_{i}}{\partial x_{j}},\ \left(k1\, x_{j}\right) . \left(k2\, c^{j}\right),\ x_{j} \otimes c^{j}\right\}$

Restore old settings...

```
In[27]:= ClearTensorShortcuts[{{x, c}, 1}, {{a, b}, 2}]
```

```
In[28]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

UseCoordinates

- `UseCoordinates[{r, θ, ϕ ...}, coord : x, flavor : Identity][expr]` replaces the base coordinate positions in expr by the list of symbols. The optional arguments coord and flavor give the coordinate label, if not x, and the index flavor, if not plain.
- `UseCoordinates[coord : x, flavor : Identity][expr]` uses the index symbols established with `DeclareBaseIndices`, in which case they should be symbols and not integers.

UseCoordinates is for the purposes of obtaining a final expression in terms of familiar variable symbols.

UseCoordinates can also be used to substitute specific numerical values or expressions on a one time basis.

Another method is to set tensor values for the coordinate vectors and then substitute them.

See also: `CoordinatesToTensors`, `DeclareBaseIndices`, `EinsteinArgument`.

Examples

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

Save settings

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

In[7]:= DefineTensorShortcuts[{{x, y, dx}, 1}, {{g, η}, 2}]
```

Here is a simple case of base coordinate positions being replace by coordinate symbols.

```
In[8]:= xu[i]
        % // EinsteinArray[]
        % // UseCoordinates[{x, y, z}]

Out[8]= x^i

Out[9]= {x^1, x^2, x^3}

Out[10]= {x, y, z}
```

For flavored indices, and for coordinate symbols other than x, we have to include the optional arguments.

```
In[11]:= yu[i] // ToFlavor[red]
         % // EinsteinArray[]
         % // UseCoordinates[{x, y, z}, y, red]

Out[11]= y^i

Out[12]= {y^1, y^2, y^3}

Out[13]= {x, y, z}
```

If we have set symbols for the base indices, then we can use them directly.

```
In[14]:= DeclareBaseIndices[{x, y, z}]
         xu[i] // EinsteinArray[]
         % // UseCoordinates[]

Out[15]= {x^x, x^y, x^z}

Out[16]= {x, y, z}
```

Here is a line element in spherical coordinates. The base index symbols are specified as {r, θ, ϕ}.

```
In[17]:= DeclareBaseIndices[{r, θ, ϕ}]
         metric = ( 1      0              0
                  ( 0   xu[r]^2           0
                  ( 0      0     (xu[r] Sin[xu[θ]])^2 );
         SetTensorValues[gdd[a, b], metric]

In[20]:= gdd[i, j] dxu[i] dxu[j]
         % // EinsteinSum[]
         % // UseCoordinates[] // UseCoordinates[{dr, dθ, dϕ}, dx]

Out[20]= dx^i dx^j g_i j

Out[21]= (dx^r)^2 + (dx^θ)^2 (x^r)^2 + Sin[x^θ]^2 (dx^ϕ)^2 (x^r)^2

Out[22]= dr^2 + dθ^2 r^2 + dϕ^2 r^2 Sin[θ]^2
```

The invariant interval in special relativity

```
In[23]:= DeclareBaseIndices[{0, 1, 2, 3}]
         metric = DiagonalMatrix[{1, -1, -1, -1}];
         SetTensorValues[ηdd[a, b], metric]

In[26]:= ηdd[i, j] dxu[i] dxu[j]
         % // EinsteinSum[]
         dτ^2 == % // UseCoordinates[{dt, dx, dy, dz}, dx]

Out[26]= dx^i dx^j η_i j

Out[27]= (dx^0)^2 - (dx^1)^2 - (dx^2)^2 - (dx^3)^2

Out[28]= dτ^2 == dt^2 - dx^2 - dy^2 - dz^2
```

UseCoordinates can be used to substitute specific values into an expression involving coordinates. This is convenient if you want to do it on a one time basis.

```
In[29]:= DeclareBaseIndices[{1, 2, 3}]
         f[x_, y_, z_] := x y Cos[z]
         f[xu[a]]
         % // EinsteinArgument[x]
         % // UseCoordinates[{1, 2, π}]
         Clear[f]

Out[31]= f[x^a]

Out[32]= Cos[x^3] x^1 x^2

Out[33]= -2
```

Here UseCoordinates is used to substitute the parametrization of a helix.

```
In[35]:= xu[a]
         % // EinsteinArray[]
         % // UseCoordinates[{Cos[t], Sin[t], t}]

Out[35]= x^a

Out[36]= {x^1, x^2, x^3}

Out[37]= {Cos[t], Sin[t], t}
```

In the following, UseCoordinates is used to calculate the Jacobian of a coordinate transformation {X == x, Y == x + y}.

```
In[38]:= DeclareBaseIndices[{1, 2}]
         PartialD[{x, δ, g, Γ}][xu[red@a], xu[b]]
         % // EinsteinArray[]
         % // UseCoordinates[{x, y}] // UseCoordinates[{x, x + y}, x, red]

Out[39]= ∂x^a/∂x^b

Out[40]= {{∂x^1/∂x^1, ∂x^1/∂x^2}, {∂x^2/∂x^1, ∂x^2/∂x^2}}

Out[41]= {{1, 0}, {1, 1}}
```

UseCoordinates can be used with multiple base indices.

```
In[42]:= DeclareBaseIndices[{x, y, z}, {red, {ρ, θ, ϕ}}]

In[43]:= xu[a] // EinsteinArray[]
         % // UseCoordinates[]

Out[43]= {x^x, x^y, x^z}

Out[44]= {x, y, z}

In[45]:= xu[red@a] // EinsteinArray[]
         % // UseCoordinates[x, red]

Out[45]= {x^ρ, x^θ, x^ϕ}

Out[46]= {ρ, θ, ϕ}
```

Restore settings

```
In[47]:= ClearTensorShortcuts[{{x, y, dx}, 1}, {{g, η}, 2}]

In[48]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

UsePartialDChainRule

- `UsePartialDChainRule[`*coordtensor*`][`*expr*`]` expands partial derivatives over coordtensor using the chain rule.
- `UsePartialDChainRule[`*diffvar1, diffvar2*`][`*expr*`]` expands single or multiple partial derivatives with repspect to diffvar2 over diffvar1 using the chain rule.

Replacements will only occur on partial derivatives in the expanded form.

See also: `PartialD`, `ExpandPartialD`, `TotalD`, `UseTotalDChainRule`.

Examples

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

Save settings.

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

Define the tensor shortcuts and derivative labels.

```
In[6]:= DefineTensorShortcuts[{{x, T}, 1}, {{S}, 2}]
        labs = {x, δ, g, Γ};
```

Here a partial derivative is expanded using the chain rule with respect to a flavored coordinate system.

```
In[8]:= Tu[i]
        PartialD[labs][%, xu[j]]
        % // UsePartialDChainRule[xu[red@a]]
```

```
Out[8]= T^i
```

$$\text{Out[9]=}\quad \frac{\partial T^{i}}{\partial x^{j}}$$

$$\text{Out[10]=}\quad \frac{\partial T^{i}}{\partial x^{a}}\,\frac{\partial x^{a}}{\partial x^{j}}$$

The command works on all expanded partial derivatives in the expression. Use `MapAt` to restrict to specific parts of an expression.

```
In[11]:= Tu[i] Sdd[i, j]
         PartialD[labs][%, xu[k]]
         % // UsePartialDChainRule[xu[red@a]]
```

Out[11]= $S_{i\,j}\,T^{i}$

Out[12]= $T^{i}\,\frac{\partial S_{i\,j}}{\partial x^{k}} + S_{i\,j}\,\frac{\partial T^{i}}{\partial x^{k}}$

Out[13]= $T^{i}\,\frac{\partial S_{i\,j}}{\partial x^{a}}\,\frac{\partial x^{a}}{\partial x^{k}} + S_{i\,j}\,\frac{\partial T^{i}}{\partial x^{a}}\,\frac{\partial x^{a}}{\partial x^{k}}$

Using the chain rule with the same index flavor and coordinate simply returns to the original expression.

```
In[14]:= Tu[i] Sdd[i, j]
         PartialD[labs][%, xu[k]]
         % // UsePartialDChainRule[xu[a]]
         % // KroneckerAbsorb[δ]
```

Out[14]= $S_{i\,j}\,T^{i}$

Out[15]= $T^{i}\,\frac{\partial S_{i\,j}}{\partial x^{k}} + S_{i\,j}\,\frac{\partial T^{i}}{\partial x^{k}}$

Out[16]= $T^{i}\,\delta^{a}{}_{k}\,\frac{\partial S_{i\,j}}{\partial x^{a}} + S_{i\,j}\,\delta^{a}{}_{k}\,\frac{\partial T^{i}}{\partial x^{a}}$

Out[17]= $T^{i}\,\frac{\partial S_{i\,j}}{\partial x^{k}} + S_{i\,j}\,\frac{\partial T^{i}}{\partial x^{k}}$

Higher order derivatives can be expanded using the second form of the command.

```
In[18]:= PartialD[labs][Sdd[i, j], {xu[a], xu[red@b]}]
         % // UsePartialDChainRule[xu[c], xu[red@b]]
```

Out[18]= $\frac{\partial^{2} S_{i\,j}}{\partial x^{a}\,\partial x^{b}}$

Out[19]= $\frac{\partial^{2} S_{i\,j}}{\partial x^{a}\,\partial x^{c}}\,\frac{\partial x^{c}}{\partial x^{b}}$

Restore settings.

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

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

UseTotalDChainRule

- `UseTotalDChainRule[`*coordtensor, labels*`][`*expr*`]` expands total derivatives over coordtensor using the chain rule. The tensor labels are used in the generated partial derivative.

The command works only of first order total derivatives.

See also: `TotalD`, `ExpandTotalD`, `PartialD`, `UsePartialDChainRule`.

Examples

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

Save settings.

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

Define the tensor shortcuts and derivative labels.

```
In[6]:= DefineTensorShortcuts[{{x, T}, 1}, {{S}, 2}]
        labs = {x, δ, g, Γ};
```

Here a total derivative is expanded using the chain rule with respect to a flavored coordinate system.

```
In[8]:= Tu[i]
        TotalD[%, t]
        % // UseTotalDChainRule[xu[red@a], labs]
```

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

$$\text{Out[9]=}\ \frac{dT^{i}}{dt}$$

$$\text{Out[10]=}\ \frac{dx^{a}}{dt}\ \frac{\partial T^{i}}{\partial x^{a}}$$

The command works on all first order total derivatives in the expression. Use `MapAt` to restrict to specific parts of an expression.

```
In[11]:= Tu[i] Sdd[i, j]
         TotalD[%, t]
         % // UseTotalDChainRule[xu[red@a], labs]
```

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

$$\text{Out[12]=}\ T^{i}\ \frac{dS_{i\,j}}{dt} + S_{i\,j}\ \frac{dT^{i}}{dt}$$

$$\text{Out[13]=}\ T^{i}\ \frac{dx^{a}}{dt}\ \frac{\partial S_{i\,j}}{\partial x^{a}} + S_{i\,j}\ \frac{dx^{a}}{dt}\ \frac{\partial T^{i}}{\partial x^{a}}$$

Using the chain rule with the same flavor and coordinate is the same as using `ExpandTotalD`.

```
In[14]:= Tu[i] Sdd[i, j]
         TotalD[%, t]
         % // UseTotalDChainRule[xu[a], labs]
```

Out[14]= $S_{i\,j}\,T^{i}$

Out[15]= $T^{i}\,\frac{dS_{i\,j}}{dt} + S_{i\,j}\,\frac{dT^{i}}{dt}$

Out[16]= $T^{i}\,\frac{dx^{a}}{dt}\,\frac{\partial S_{i\,j}}{\partial x^{a}} + S_{i\,j}\,\frac{dx^{a}}{dt}\,\frac{\partial T^{i}}{\partial x^{a}}$

```
In[17]:= Tu[i] Sdd[i, j]
         TotalD[%, t]
         % // ExpandTotalD[labs, a]
```

Out[17]= $S_{i\,j}\,T^{i}$

Out[18]= $T^{i}\,\frac{dS_{i\,j}}{dt} + S_{i\,j}\,\frac{dT^{i}}{dt}$

Out[19]= $T^{i}\,\frac{dx^{a}}{dt}\,\frac{\partial S_{i\,j}}{\partial x^{a}} + S_{i\,j}\,\frac{dx^{a}}{dt}\,\frac{\partial T^{i}}{\partial x^{a}}$

Restore settings.

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

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

Void

■ `Void` is used to fill the empty slot in an up/down pair of indices in a Tensor.

Each index slot (up/down pair) must have one index and one Void.

See also: `Tensor`, `DefineTensorShortcuts`.

Examples

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

In[2]:= Tensor[T, {i, Void, k}, {Void, j, Void}]

Out[2]= T^i _j^k
```