Tensor

- `Tensor[`*a, upindices, downindices*`]` represents a tensor or indexed object with label a and the given up and down indices
- `Tensor[`$\emptyset$`]` represents a scalar tensor

Tensorial uses the heading Tensor even though the objects in question may not be true tensors, for example coordinates or Christoffel symbols or transformation matrices. We debated using the heading IndexedObject but decided to stay with Tensor.

The up and down indices are specified in lists.

The number of up and down indices must be equal.

Missing indices are specified by the symbol `Void`. Each index slot (up/down pair) must have one index and one Void.

Indices can have flavors corresponding to different coordinate systems or reference frames.

Tensors can be more easily entered by defining and using tensor shortcuts. In normal use one might never use or see the Tensor heading.

Tensors are formatted on output.

See also: `TensorLabelFormat`, `Void`, `SetDerivativeSymbols`, `DefineTensorShortcuts`, `SetTensorValues`, `DeclareBaseIndices`, `DeclareIndexFlavor`.

Examples

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

This is a simple contravariant tensor.

```
In[2]:= Tensor[v, {i}, {Void}]
```

```
Out[2]= v^i
```

A more complex tensor is...

```
In[3]:= Tensor[T, {i, Void, Void}, {Void, j, k}]
```

```
Out[3]= T^i
        j k
```

Tensors are more easily entered using tensor shortcuts. See `DefineTensorShortcuts`.

TensorSimplify

- `TensorSimplify[`*expr*`]` will attempt to simplify a tensor expression by applying declared symmetires and reindexing terms with similar patterns.

TensorSimplify is automatically mapped over lists and equations and expressions are automatically expanded.

TensorSimplify will work best when any tensor symmetries are predeclared with TensorSymmetry definitions and DeclarePattern-Symmetries.

TensorSimplify works only on terms that contain dummy indices.

TensorSimplify does not perform any dummy UpDownSwaps. Using UpDownAdjust and/or UpDownSwap on selected terms may allow further simplification.

Sometimes TensorSimplify will do further simplification if applied several times.

TensorSimplify is like Simplify. It will do a lot but not everything. Further tailored simplifications may be possible.

Further, or restricted simplifications may be performed with UpDownSwap, IndexChange, SymmetrizeSlots, SymmetrizePattern and SimplifyTensorSum, all applied to specific terms using MapAt, MapLevelParts or MapLevelPatterns.

See also: `SimplifyTensorSum`, `UpDownAdjust`, `UpDownSwap`, `IndexChange`, `SymmetrizeSlots`, `Symmetrize-Pattern`, `NondependentPartialD`, `MapLevelParts`, `MapLevelPatterns`, `TensorSymmetry`, `DeclarePattern-Symmetries`.

Examples

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

Save old settings and declare an index flavor.

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

```
In[5]:= DefineTensorShortcuts[{{x, λ, p}, 1}, {{g, R, T, S}, 2}, {T, 3}, {R, 4}]
        DeclareIndexFlavor[{red, Red}]
        labs = {x, δ, g, Γ};
```

This declares tensor symmetries used in the examples.

```
In[8]:= TensorSymmetry[g, 2] = Symmetric[1, 2];
        TensorSymmetry[Γ, 3] = Symmetric[2, 3];
```

In the following, symmetry and reindexing are used to simplify the expression.

```
In[10]:= 3 gdd[b, a] pu[b] + Sin[θ] gdd[a, c] pu[c] +
          Tuud[b, c, a] gdd[b, c] + Tuud[c, d, a] gdd[c, d]
         % // TensorSimplify
```

$$\text{Out[10]= } 3\, g_{b\,a}\, p^{b} + \text{Sin}[\theta]\, g_{a\,c}\, p^{c} + g_{b\,c}\, T^{b\,c}{}_{a} + g_{c\,d}\, T^{c\,d}{}_{a}$$

$$\text{Out[11]= } (3 + \text{Sin}[\theta])\, g_{a\,b}\, p^{b} + 2\, g_{b\,c}\, T^{b\,c}{}_{a}$$

```
In[12]:= a gdd[a, b] + b gdd[b, a]
         % // TensorSimplify // Factor
```

$$\text{Out[12]= } a\, g_{a\,b} + b\, g_{b\,a}$$

$$\text{Out[13]= } (a + b)\, g_{a\,b}$$

Covariant differentiation is not commutative. In the following we calculate the commutator of a double differentiation acting on a vector field.

```
In[14]:= SetCovariantDisplay["DelMode"]
```

```
In[15]:= Print["Covariant differentiation is not commutative"]
         (CovariantD[λu[a], {b, c}] ≠ CovariantD[λu[a], {c, b}]) // ToFlavor[red]
         # - Part[%, 2] & /@ %
         Print["Expand"]
         %% // ExpandCovariantD[{x, δ, g, Γ}, red /@ {d, e}]
         Print["Use TensorSimplify and factor"]
         %% // TensorSimplify
         MapAt[Factor, %, 1]
```

Covariant differentiation is not commutative

$$\text{Out[16]= } \nabla_{b\,c}\lambda^{a} \neq \nabla_{c\,b}\lambda^{a}$$

$$\text{Out[17]= } \nabla_{b\,c}\lambda^{a} - \nabla_{c\,b}\lambda^{a} \neq 0$$

Expand

$$\text{Out[19]= } \lambda^{d}\frac{\partial \Gamma^{a}{}_{b\,d}}{\partial x^{c}} - \lambda^{d}\frac{\partial \Gamma^{a}{}_{c\,d}}{\partial x^{b}} - \frac{\partial^{2}\lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2}\lambda^{a}}{\partial x^{c}\,\partial x^{b}} + \Gamma^{e}{}_{b\,c}\left(\Gamma^{a}{}_{e\,d}\,\lambda^{d} + \frac{\partial \lambda^{a}}{\partial x^{e}}\right) - \Gamma^{e}{}_{c\,b}\left(\Gamma^{a}{}_{e\,d}\,\lambda^{d} + \frac{\partial \lambda^{a}}{\partial x^{e}}\right) - \Gamma^{a}{}_{c\,d}\frac{\partial \lambda^{d}}{\partial x^{b}} + \Gamma^{a}{}_{b\,d}\frac{\partial \lambda^{d}}{\partial x^{c}} + \Gamma^{a}{}_{c\,e}\left(\Gamma^{e}{}_{b\,d}\,\lambda^{d} + \frac{\partial \lambda^{e}}{\partial x^{b}}\right) - \Gamma^{a}{}_{b\,e}\left(\Gamma^{e}{}_{c\,d}\,\lambda^{d} + \frac{\partial \lambda^{e}}{\partial x^{c}}\right) \neq 0$$

Use TensorSimplify and factor

$$\text{Out[21]= } \Gamma^{a}{}_{c\,e}\,\Gamma^{e}{}_{b\,d}\,\lambda^{d} - \Gamma^{a}{}_{b\,e}\,\Gamma^{e}{}_{c\,d}\,\lambda^{d} + \lambda^{d}\frac{\partial \Gamma^{a}{}_{b\,d}}{\partial x^{c}} - \lambda^{d}\frac{\partial \Gamma^{a}{}_{c\,d}}{\partial x^{b}} - \frac{\partial^{2}\lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2}\lambda^{a}}{\partial x^{c}\,\partial x^{b}} \neq 0$$

$$\text{Out[22]= } \Gamma^{a}{}_{c\,e}\,\Gamma^{e}{}_{b\,d}\,\lambda^{d} - \Gamma^{a}{}_{b\,e}\,\Gamma^{e}{}_{c\,d}\,\lambda^{d} + \lambda^{d}\frac{\partial \Gamma^{a}{}_{b\,d}}{\partial x^{c}} - \lambda^{d}\frac{\partial \Gamma^{a}{}_{c\,d}}{\partial x^{b}} - \frac{\partial^{2}\lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2}\lambda^{a}}{\partial x^{c}\,\partial x^{b}} \neq 0$$

The expression in brackets gives the elements of the Riemann curvature tensor.

The following expands the covariant derivative of a scalar quantity. This generates Christoffel correction terms for the vectors, but they cancel out when simplified leaving an ordinary partial derivative. UpDownAdjust simplifies the last two terms.

```
In[23]:= Tensor[ϕ] + pu[a] pd[a]
         CovariantD[%, b]
         % // ExpandCovariantD[labs, c]
         % // TensorSimplify
         % // UpDownAdjust
```

$$\text{Out[23]=}\quad \phi + p^a\, p_a$$

$$\text{Out[24]=}\quad \phi_{,b} + \nabla_b p_a\, p^a + \nabla_b p^a\, p_a$$

$$\text{Out[25]=}\quad \phi_{,b} + p_a \left(p^c\, \Gamma^a{}_{bc} + \frac{\partial p^a}{\partial x^b}\right) + p^a \left(-p_c\, \Gamma^c{}_{ba} + \frac{\partial p_a}{\partial x^b}\right)$$

$$\text{Out[26]=}\quad \phi_{,b} + p^c\, p_a\, \Gamma^a{}_{bc} - p^a\, p_c\, \Gamma^c{}_{ab} + p_a \frac{\partial p^a}{\partial x^b} + p^a \frac{\partial p_a}{\partial x^b}$$

$$\text{Out[27]=}\quad \phi_{,b} + p^a\, p^c\, \Gamma_{abc} - p^a\, p^c\, \Gamma_{cab} + 2\, p^a \frac{\partial p_a}{\partial x^b}$$

With a second order derivative we have to use UpDownSwap to complete a simplification.

```
In[28]:= Tensor[ϕ] + pu[a] pd[a]
         Print["Covariant derivative expanded"]
         CovariantD[%%, {b, c}]
         % // ExpandCovariantD[labs, {d, e}]
         Print["TensorSimplify"]
         %% // TensorSimplify
         Print["Using UpDownAdjust consolidates some terms but
            misses two terms involving products of partial derivatives."]
         %% // UpDownAdjust
         Print["Those two terms are simplified by using a specific UpDownSwap."]
         MapAt[UpDownSwap[a], %%, 7]
```

$$\text{Out[28]=}\quad \phi + p^a\, p_a$$

```
Covariant derivative expanded
```

$$\text{Out[30]=}\quad \nabla_{bc}\phi + \nabla_c p^a\, \nabla_b p_a + \nabla_b p^a\, \nabla_c p_a + \nabla_{bc} p_a\, p^a + \nabla_{bc} p^a\, p_a$$

$$\begin{aligned}\text{Out[31]=}\quad & \frac{\partial^2 \phi}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb} \frac{\partial \phi}{\partial x^e} + \left(p^d\, \Gamma^a{}_{cd} + \frac{\partial p^a}{\partial x^c}\right) \left(-p_e\, \Gamma^e{}_{ba} + \frac{\partial p_a}{\partial x^b}\right) + \left(p^d\, \Gamma^a{}_{bd} + \frac{\partial p^a}{\partial x^b}\right) \left(-p_e\, \Gamma^e{}_{ca} + \frac{\partial p_a}{\partial x^c}\right) + \\ & p_a \left(\frac{\partial^2 p^a}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb} \left(p^d\, \Gamma^a{}_{ed} + \frac{\partial p^a}{\partial x^e}\right) + \Gamma^a{}_{bd} \frac{\partial p^d}{\partial x^c} + \Gamma^a{}_{ce} \left(p^d\, \Gamma^e{}_{bd} + \frac{\partial p^e}{\partial x^b}\right) + p^d \frac{\partial \Gamma^a{}_{bd}}{\partial x^c}\right) + \\ & p^a \left(\frac{\partial^2 p_a}{\partial x^c\, \partial x^b} - \Gamma^e{}_{cb} \left(-p_d\, \Gamma^d{}_{ea} + \frac{\partial p_a}{\partial x^e}\right) - \Gamma^d{}_{ba} \frac{\partial p_d}{\partial x^c} - \Gamma^e{}_{ca} \left(-p_d\, \Gamma^d{}_{be} + \frac{\partial p_e}{\partial x^b}\right) - p_d \frac{\partial \Gamma^d{}_{ba}}{\partial x^c}\right)\end{aligned}$$

```
TensorSimplify
```

In the following we declare a pattern symmetry. FullTensorSimplify applies the symmetry and reindexing to simplify.

```
In[38]:= DeclarePatternSymmetries[Sud[_, b_] Tdd[_, c_], {{1, {b, c}}}]
```

```
In[39]:= Sud[a, b] Tdd[a, c] + Sud[d, c] Tdd[d, b]
         % // TensorSimplify
```

The contraction of symmetrical slots with antisymmetrical slots is zero but is not automatically detected. It can be implemented with a `PatternSymmetry`.

```
In[41]:= DeclarePatternSymmetries[Ruu[a_, b_] Rdddd[a_, b_, _, _], {{0, {a, b}}}]
```

```
In[42]:= Ruu[a, b] Rdddd[a, b, c, d]
         % // TensorSimplify
```

Restore old settings...

```
In[44]:= ClearTensorShortcuts[{{x, λ, p}, 1}, {{g, R, T, S}, 2}, {T, 3}, {R, 4}]

In[45]:= SetCovariantDisplay["SemicolonMode"]

In[46]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

TensorSymmetry

- `TensorSymmetry[`*label, order*`] = symmetry` will store the symmetry specifications for tensors with the given label and order (number of slots).

The symmetry is specified with the following syntax:

```
symmetrytype := Symmetric | AntiSymmetric | List

symmetry := symmetrytype[symmetryarg ..]

symmetryarg := (slot number) | symmetry
```

`Symmetric` means all permutations of the slots are allowed.

`AntiSymmetric` means all permutations of the slots are allowed, weighted by the signature of the permutation.

`List` means there are no further symmetries in the slots.

Examples of symmetries are:

`TensorSymmetry[g, 2] = Symmetric[1, 2]` - second order g tensor symmetric in the two slots.

`TensorSymmetry[R, 4] = Symmetric[AntiSymmetric[1, 2], AntiSymmetric[3, 4]]`

- Riemann tensor antisymmetric is slots 1 and 2 and again in slots 3 and 4, and symmetric under the the interchange of the first pair of slots with the last pair of slots.

`TensorSymmetry[T, 4] = {AntiSymmetric[1, 2], AntiSymmetric[3, 4]}` - Antisymmetric in the two pairs of slots but with no further symmetries.

`TensorSymmetry[S, 4] = Symmetric[{1, 3}, {2, 4}]` - Symmetric under interchange of the two pairs of slots.

Processing of a symmetry:

When a symmetry is present the associated slots will be permuted to their lowest lexicographic order.

Repeated dummy indices will put the up slot first.

The weighting of the reordered tensor will be

+1 for symmetric slots

the signature of the permutation for antisymmetric slots

0 if the slots are antisymmetric and they contain a dummy index.

The tensor symmetries may be applied by using `SymmetrizeSlots[]` on an expression.

A tensor symmetry can be cleared by the usual Mathematical command: `TensorSymmetry[label, order] =.`.

A symmetry for a specific label and order can be reset without clearing.

See also: `SymmetrizeSlots`, `TensorSimplify`, `DeclarePatternSymmetries`, `Symmetric`, `AntiSymmetric`, `IndexChange`.

Examples

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

```
In[2]:= DefineTensorShortcuts[{{R, T}, 2}, {ω, 3}, {{R, T}, 4}, {T, 6}]
```

The following statements define T as a symmetric tensor and ω as antisymmetric in the first two slots.

```
In[3]:= TensorSymmetry[T, 2] = Symmetric[1, 2];
        TensorSymmetry[ω, 3] = AntiSymmetric[1, 2];
```

We can then use these symmetries to order or simplify various expressions without having to specify the symmetries each time.

```
In[5]:= ωudd[a, a, b] Tuu[b, c]
        % // SymmetrizeSlots[]
```

$$\text{Out[5]= } T^{b\,c}\,\omega^{a}{}_{a\,b}$$

$$\text{Out[6]= } 0$$

```
In[7]:= ωudd[a, b, a] Tuu[c, b]
        % // SymmetrizeSlots[]
```

$$\text{Out[7]= } T^{c\,b}\,\omega^{a}{}_{b\,a}$$

$$\text{Out[8]= } T^{b\,c}\,\omega^{a}{}_{b\,a}$$

```
In[9]:= ωudd[b, a, b] Tuu[c, a]
        % // SymmetrizeSlots[]
```

$$\text{Out[9]= } T^{c\,a}\,\omega^{b}{}_{a\,b}$$

$$\text{Out[10]= } -T^{a\,c}\,\omega_{a}{}^{b}{}_{b}$$

```
In[11]:= ωdud[a, b, b] Tuu[c, a] + ωudd[b, a, b] Tuu[c, a]
         % // SymmetrizeSlots[]
```

$$\text{Out[11]= } T^{c\,a}\,\omega^{b}{}_{a\,b} + T^{c\,a}\,\omega_{a}{}^{b}{}_{b}$$

$$\text{Out[12]= } 0$$

The symmetries are cleared by...

```
In[13]:= TensorSymmetry[T, 2] =.
         TensorSymmetry[ω, 3] =.
```

The last expression above no longer simplifies.

```
In[15]:= ωdud[a, b, b] Tuu[c, a] + ωudd[b, a, b] Tuu[c, a]
         % // SymmetrizeSlots[]
```

$$\text{Out[15]= } T^{c\,a}\,\omega^{b}{}_{a\,b} + T^{c\,a}\,\omega_{a}{}^{b}{}_{b}$$

$$\text{Out[16]= } T^{c\,a}\,\omega^{b}{}_{a\,b} + T^{c\,a}\,\omega_{a}{}^{b}{}_{b}$$

The following are some further symmetries.

```
In[17]:= TensorSymmetry[R, 4] = Symmetric[AntiSymmetric[1, 2], AntiSymmetric[3, 4]];
         TensorSymmetry[T, 4] = {AntiSymmetric[1, 2], AntiSymmetric[3, 4]};
         TensorSymmetry[R, 2] = Symmetric[1, 2];
         TensorSymmetry[T, 2] = AntiSymmetric[1, 2];
```

```
In[21]:= testlist = {Ruddd[b, a, d, c], Ruddd[b, a, c, d], Ruddd[a, b, d, c],
            Ruddd[a, a, d, c], Rddud[b, a, c, c], Ruddd[a, b, c, a], Rdddd[c, d, a, b]};
         TensorSymmetry[R, 4]
         Thread[testlist → SymmetrizeSlots[][testlist]] // TableForm

Out[22]= Symmetric[AntiSymmetric[1, 2], AntiSymmetric[3, 4]]
```

Out[23]//TableForm=

$$R^{b}{}_{adc} \to R_{a}{}^{b}{}_{cd}$$
$$R^{b}{}_{acd} \to -R_{a}{}^{b}{}_{cd}$$
$$R^{a}{}_{bdc} \to -R^{a}{}_{bcd}$$
$$R^{a}{}_{adc} \to 0$$
$$R_{ba}{}^{c}{}_{c} \to 0$$
$$R^{a}{}_{bca} \to -R^{a}{}_{bac}$$
$$R_{cdab} \to R_{abcd}$$

```
In[24]:= testlist = {Tuddd[b, a, d, c], Tuddd[b, a, c, d], Tuddd[a, b, d, c], Tuddd[a, a, d, c],
            Tddud[b, a, c, c], Tuddd[a, b, c, a], Tdddd[d, c, b, a], Tdddd[c, d, b, a]};
         TensorSymmetry[T, 4]
         Thread[testlist → SymmetrizeSlots[][testlist]] // TableForm

Out[25]= {AntiSymmetric[1, 2], AntiSymmetric[3, 4]}
```

Out[26]//TableForm=

$$T^{b}{}_{adc} \to T_{a}{}^{b}{}_{cd}$$
$$T^{b}{}_{acd} \to -T_{a}{}^{b}{}_{cd}$$
$$T^{a}{}_{bdc} \to -T^{a}{}_{bcd}$$
$$T^{a}{}_{adc} \to 0$$
$$T_{ba}{}^{c}{}_{c} \to 0$$
$$T^{a}{}_{bca} \to -T^{a}{}_{bac}$$
$$T_{dcba} \to T_{cdab}$$
$$T_{cdba} \to -T_{cdab}$$

```
In[27]:= testlist = {Tuddd[b, a, d, c], Tuddd[b, a, c, d], Tuddd[a, b, d, c], Tuddd[a, a, d, c],
            Tddud[b, a, c, c], Tuddd[a, b, c, a], Tdddd[d, c, b, a], Tdddd[c, d, b, a]};
         TensorSymmetry[T, 4] = AntiSymmetric[{1, 2}, {3, 4}]
         Thread[testlist → SymmetrizeSlots[][testlist]] // TableForm

Out[28]= AntiSymmetric[{1, 2}, {3, 4}]
```

Out[29]//TableForm=

$$T^{b}{}_{adc} \to T^{b}{}_{adc}$$
$$T^{b}{}_{acd} \to T^{b}{}_{acd}$$
$$T^{a}{}_{bdc} \to T^{a}{}_{bdc}$$
$$T^{a}{}_{adc} \to T^{a}{}_{adc}$$
$$T_{ba}{}^{c}{}_{c} \to T_{ba}{}^{c}{}_{c}$$
$$T^{a}{}_{bca} \to 0$$
$$T_{dcba} \to -T_{badc}$$
$$T_{cdba} \to -T_{bacd}$$

```
In[30]:= testlist = {Rdd[a, b], Rdd[b, a], Rud[a, b], Rud[b, a], Rud[a, a], Rdu[a, a]};
         TensorSymmetry[R, 2]
         Thread[testlist → SymmetrizeSlots[][testlist]] // TableForm
```

```
Out[31]= Symmetric[1, 2]
```

Out[32]//TableForm=

$$R_{a\,b} \to R_{a\,b}$$
$$R_{b\,a} \to R_{a\,b}$$
$$R^{a}{}_{b} \to R^{a}{}_{b}$$
$$R^{b}{}_{a} \to R_{a}{}^{b}$$
$$R^{a}{}_{a} \to R^{a}{}_{a}$$
$$R_{a}{}^{a} \to R^{a}{}_{a}$$

```
In[33]:= testlist = {Tdd[a, b], Tdd[b, a], Tud[a, b], Tud[b, a], Tud[a, a], Tdu[a, a]};
         TensorSymmetry[T, 2]
         Thread[testlist → SymmetrizeSlots[][testlist]] // TableForm
```

```
Out[34]= AntiSymmetric[1, 2]
```

Out[35]//TableForm=

$$T_{a\,b} \to T_{a\,b}$$
$$T_{b\,a} \to -T_{a\,b}$$
$$T^{a}{}_{b} \to T^{a}{}_{b}$$
$$T^{b}{}_{a} \to -T_{a}{}^{b}$$
$$T^{a}{}_{a} \to 0$$
$$T_{a}{}^{a} \to 0$$

```
In[36]:= testlist =
           {Tdddddd[b, a, d, c, f, e], Tdddddd[f, e, b, a, d, c], Tdddddu[f, e, b, a, c, c],
            Tdduddd[f, e, a, a, d, c], Tuddddd[a, e, b, a, d, c], Tuddddd[a, e, f, e, a, c]};
         TensorSymmetry[T, 6] = AntiSymmetric[{1, 2}, {3, 4}, {5, 6}]
         Thread[testlist → SymmetrizeSlots[][testlist]] // TableForm
```

```
Out[37]= AntiSymmetric[{1, 2}, {3, 4}, {5, 6}]
```

Out[38]//TableForm=

$$T_{b\,a\,d\,c\,f\,e} \to T_{b\,a\,d\,c\,f\,e}$$
$$T_{f\,e\,b\,a\,d\,c} \to T_{b\,a\,d\,c\,f\,e}$$
$$T_{f\,e\,b\,a\,c}{}^{c} \to T_{b\,a\,c}{}^{c}{}_{f\,e}$$
$$T_{f\,e}{}^{a}{}_{a\,d\,c} \to T^{a}{}_{a\,d\,c\,f\,e}$$
$$T^{a}{}_{e\,b\,a\,d\,c} \to 0$$
$$T^{a}{}_{e\,f\,e\,a\,c} \to 0$$

```
In[39]:= ClearTensorShortcuts[{{R, T}, 2}, {ω, 3}, {{R, T}, 4}, {T, 6}]
```

TensorValueRules

- `TensorValueRules[label]` gives the existing substitution rules for tensors based on label.
- `TensorValueRules[label1, label2, ...]` concatenates the substitution rules for several labels.

TensorValueRules are useful when you don't want automatic substitution of component values.

If no rules have ever been set for a label `TensorValueRules[label]` returns unevaluated.

See also: `SetTensorValueRules`, `SetTensorValues`, `ClearTensorValues`, `Tensor`, `UseCoordinates`.

Examples

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

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

Here we create TensorValueRules for an x and a v vector...

```
In[3]:= SetTensorValueRules[xu[i], {Cos[t], Sin[t], 2 t}]
        TensorValueRules[x]

Out[4]= {x^1 → Cos[t], x^2 → Sin[t], x^3 → 2 t}

In[5]:= SetTensorValueRules[vd[i], {t, t^2, t^3}]
        TensorValueRules[v]

Out[6]= {v_1 → t, v_2 → t^2, v_3 → t^3}
```

We can then substitute the values into an expanded expression.

```
In[7]:= xu[i] vd[i]
        % // EinsteinSum[]
        % /. TensorValueRules[x, v]

Out[7]= v_i x^i

Out[8]= v_1 x^1 + v_2 x^2 + v_3 x^3

Out[9]= 2 t^4 + t Cos[t] + t^2 Sin[t]

In[10]:= ClearTensorValues /@ {xu[i], vd[i]};
         ClearTensorShortcuts[{x, v}, 1]
```

ToArrayValues

- `ToArrayValues[][`*expr*`]` will convert the expression to a vector, matrix or array by expansion and substitution of any stored values.

This provides a quick method to convert a tensor expression in index form to a *Mathematica* array.

TensorValueRules as well as tensor value definitions are used.

The expansion of arrays is done in the natural sort order of the raw indices in the expression.

Dummy indices are summed first, and then array expansion is performed.

If special sets of base indices have been associated with certain flavors of indices using `DeclareBaseIndices`, then those sets will be used with the corresponding flavors.

The expansions are always done over the full set of associated base indices, a change of usage from Version 3.0. It you wish to expand over subsets then use `EinsteinSum` and `EinsteinArray` separately.

See also: `DeclareBaseIndices`, `EinsteinSum`, `EinsteinArray`, `ArrayExpansion`, `BaseIndices`, `ContractArray`.

Examples

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

Save the settings.

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

In[6]:= DefineTensorShortcuts[{{x, y, u, v}, 1}, {{S, T}, 2}]
```

Set tensor value rules for some of the tensors.

```
In[7]:= SetTensorValueRules[xu[i], {1, 2, 3}]
        SetTensorValueRules[yu[i], {a, b, c}]
        SetTensorValueRules[Sud[i, j], {{4, 0, 1}, {0, 1, 0}, {1, 2, 0}}]
```

The `x` vector is espanded to it's array of values.

```
In[10]:= xu[i]
         % // ToArrayValues[]

Out[10]= x^i

Out[11]= {1, 2, 3}
```

In the red flavor x has a different dimension and different base indices.

```
In[12]:= xu[red@i]
         % // ToArrayValues[]
```

$$\text{Out[12]= } x^{i}$$

$$\text{Out[13]= } \{x^{A},\ x^{B}\}$$

The free indices in the following expression are j and k. The rows will correspond to increasing j and the columns to increasing k because that is the natural sort order of the free indices.

```
In[14]:= uu[red@i] vu[j] Tud[k, red@i]
         % // ToArrayValues[] // MatrixForm
```

$$\text{Out[14]= } T^{k}{}_{i}\, u^{i}\, v^{j}$$

Out[15]//MatrixForm=

$$\begin{pmatrix} T^{1}{}_{A}\, u^{A}\, v^{1} + T^{1}{}_{B}\, u^{B}\, v^{1} & T^{2}{}_{A}\, u^{A}\, v^{1} + T^{2}{}_{B}\, u^{B}\, v^{1} & T^{3}{}_{A}\, u^{A}\, v^{1} + T^{3}{}_{B}\, u^{B}\, v^{1} \\ T^{1}{}_{A}\, u^{A}\, v^{2} + T^{1}{}_{B}\, u^{B}\, v^{2} & T^{2}{}_{A}\, u^{A}\, v^{2} + T^{2}{}_{B}\, u^{B}\, v^{2} & T^{3}{}_{A}\, u^{A}\, v^{2} + T^{3}{}_{B}\, u^{B}\, v^{2} \\ T^{1}{}_{A}\, u^{A}\, v^{3} + T^{1}{}_{B}\, u^{B}\, v^{3} & T^{2}{}_{A}\, u^{A}\, v^{3} + T^{2}{}_{B}\, u^{B}\, v^{3} & T^{3}{}_{A}\, u^{A}\, v^{3} + T^{3}{}_{B}\, u^{B}\, v^{3} \end{pmatrix}$$

The following expands over the standard base indices and the values are all known.

```
In[16]:= 3 xu[i] yu[j] Sud[k, i]
         % // ToArrayValues[] // MatrixForm
```

$$\text{Out[16]= } 3\, S^{k}{}_{i}\, x^{i}\, y^{j}$$

Out[17]//MatrixForm=

$$\begin{pmatrix} 21\,a & 6\,a & 15\,a \\ 21\,b & 6\,b & 15\,b \\ 21\,c & 6\,c & 15\,c \end{pmatrix}$$

In the next statement, the free indices are a and j. So now the rows will correspond to increasing a and the columns to increasing j, the transpose of the previous case.

```
In[18]:= 3 xu[i] yu[j] Sud[a, i]
         % // ToArrayValues[] // MatrixForm
```

$$\text{Out[18]= } 3\, S^{a}{}_{i}\, x^{i}\, y^{j}$$

Out[19]//MatrixForm=

$$\begin{pmatrix} 21\,a & 21\,b & 21\,c \\ 6\,a & 6\,b & 6\,c \\ 15\,a & 15\,b & 15\,c \end{pmatrix}$$

Here we remove the values for the x vector.

```
In[20]:= ClearTensorValues[xu[i]]
         xu[i] yu[j] Sud[k, i]
         % // ToArrayValues[] // MatrixForm
```

$$\text{Out[21]=}\quad S^{k}{}_{i}\, x^{i}\, y^{j}$$

Out[22]//MatrixForm=

$$\begin{pmatrix} 4\,a\,x^1 + a\,x^3 & a\,x^2 & a\,x^1 + 2\,a\,x^2 \\ 4\,b\,x^1 + b\,x^3 & b\,x^2 & b\,x^1 + 2\,b\,x^2 \\ 4\,c\,x^1 + c\,x^3 & c\,x^2 & c\,x^1 + 2\,c\,x^2 \end{pmatrix}$$

And for the y vector.

```
In[23]:= ClearTensorValues[yu[i]]
         xu[i] yu[j] Sud[k, i]
         % // ToArrayValues[] // MatrixForm
```

$$\text{Out[24]=}\quad S^{k}{}_{i}\, x^{i}\, y^{j}$$

Out[25]//MatrixForm=

$$\begin{pmatrix} 4\,x^1\,y^1 + x^3\,y^1 & x^2\,y^1 & x^1\,y^1 + 2\,x^2\,y^1 \\ 4\,x^1\,y^2 + x^3\,y^2 & x^2\,y^2 & x^1\,y^2 + 2\,x^2\,y^2 \\ 4\,x^1\,y^3 + x^3\,y^3 & x^2\,y^3 & x^1\,y^3 + 2\,x^2\,y^3 \end{pmatrix}$$

And for S.

```
In[26]:= ClearTensorValues[Sud[i, j]]
         xu[i] yu[j] Sud[k, i]
         % // ToArrayValues[] // MatrixForm
```

$$\text{Out[27]=}\quad S^{k}{}_{i}\, x^{i}\, y^{j}$$

Out[28]//MatrixForm=

$$\begin{pmatrix} S^1{}_1\,x^1\,y^1 + S^1{}_2\,x^2\,y^1 + S^1{}_3\,x^3\,y^1 & S^2{}_1\,x^1\,y^1 + S^2{}_2\,x^2\,y^1 + S^2{}_3\,x^3\,y^1 & S^3{}_1\,x^1\,y^1 + S^3{}_2\,x^2\,y^1 + S^3{}_3\,x^3\,y^1 \\ S^1{}_1\,x^1\,y^2 + S^1{}_2\,x^2\,y^2 + S^1{}_3\,x^3\,y^2 & S^2{}_1\,x^1\,y^2 + S^2{}_2\,x^2\,y^2 + S^2{}_3\,x^3\,y^2 & S^3{}_1\,x^1\,y^2 + S^3{}_2\,x^2\,y^2 + S^3{}_3\,x^3\,y^2 \\ S^1{}_1\,x^1\,y^3 + S^1{}_2\,x^2\,y^3 + S^1{}_3\,x^3\,y^3 & S^2{}_1\,x^1\,y^3 + S^2{}_2\,x^2\,y^3 + S^2{}_3\,x^3\,y^3 & S^3{}_1\,x^1\,y^3 + S^3{}_2\,x^2\,y^3 + S^3{}_3\,x^3\,y^3 \end{pmatrix}$$

Restore settings.

```
In[29]:= ClearTensorShortcuts[{{x, y, u, v}, 1}, {{S, T}, 2}]
```

```
In[30]:= DeclareBaseIndices @@ oldindices;
         DeclareIndexFlavor @@ oldflavors;
         Clear[oldindices, oldflavors]
```

ToFlavor

- `ToFlavor[`*newflavor, oldflavor : Identity*`][expr]` will change all indices in expr in the old flavor to the new flavor.

The second argument is optional. Identity corresponds to plain unflavored indices. Identity may also be used as the newflavor.

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 `Sud[red@i, red@j]` but `ToFlavor` 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`"]
```

The following saves the current state.

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
```

This declares new base indices, flavors and some tensor shortcuts...

```
In[5]:= DeclareIndexFlavor[{red, Red}, {green, Cerulean}, {rocket, SuperStar}];
        DeclareBaseIndices[{1, 2, 3}, {red, {x, y, z}}]
        DefineTensorShortcuts[{x, 1}, {S, 2}]
```

Here is an expression and calculation with plain indices.

```
In[8]:= Sud[i, j] xu[j]
        % // EinsteinSum[]
        (step1 = % // EinsteinArray[]) // TableForm
```

```
Out[8]= S^i_j x^j
```

```
Out[9]= S^i_1 x^1 + S^i_2 x^2 + S^i_3 x^3
```

```
Out[10]//TableForm=
S^1_1 x^1 + S^1_2 x^2 + S^1_3 x^3
S^2_1 x^1 + S^2_2 x^2 + S^2_3 x^3
S^3_1 x^1 + S^3_2 x^2 + S^3_3 x^3
```

This does the same calculation using red indices. Symbolic base indices were declared for the red flavor.

```
In[11]:= Sud[i, j] xu[j] // ToFlavor[red]
         % // EinsteinSum[]
         (step2 = % // EinsteinArray[]) // TableForm
```

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

Out[12]= $S^{i}{}_{x}\, x^{x} + S^{i}{}_{y}\, x^{y} + S^{i}{}_{z}\, x^{z}$

Out[13]//TableForm=

$$S^{x}{}_{x}\, x^{x} + S^{x}{}_{y}\, x^{y} + S^{x}{}_{z}\, x^{z}$$
$$S^{y}{}_{x}\, x^{x} + S^{y}{}_{y}\, x^{y} + S^{y}{}_{z}\, x^{z}$$
$$S^{z}{}_{x}\, x^{x} + S^{z}{}_{y}\, x^{y} + S^{z}{}_{z}\, x^{z}$$

And this does it in the rocket frame...

```
In[14]:= Sud[i, j] xu[j] // ToFlavor[rocket]
         % // EinsteinSum[]
         % // EinsteinArray[] // TableForm
```

Out[14]= $S^{i^*}{}_{j^*}\, x^{j^*}$

Out[15]= $S^{i^*}{}_{1^*}\, x^{1^*} + S^{i^*}{}_{2^*}\, x^{2^*} + S^{i^*}{}_{3^*}\, x^{3^*}$

Out[16]//TableForm=

$$S^{1^*}{}_{1^*}\, x^{1^*} + S^{1^*}{}_{2^*}\, x^{2^*} + S^{1^*}{}_{3^*}\, x^{3^*}$$
$$S^{2^*}{}_{1^*}\, x^{1^*} + S^{2^*}{}_{2^*}\, x^{2^*} + S^{2^*}{}_{3^*}\, x^{3^*}$$
$$S^{3^*}{}_{1^*}\, x^{1^*} + S^{3^*}{}_{2^*}\, x^{2^*} + S^{3^*}{}_{3^*}\, x^{3^*}$$

Here we switch between various flavors, ending with an error.

```
In[17]:= Sud[i, j] xu[j]
         % // ToFlavor[red]
         % // ToFlavor[green, red]
         % // ToFlavor[rocket, green]
         % // ToFlavor[Identity, rocket]
         % // ToFlavor[blue]
```

Out[17]= $S^{i}{}_{j}\, x^{j}$

Out[18]= $S^{i}{}_{j}\, x^{j}$

Out[19]= $S^{i}{}_{j}\, x^{j}$

Out[20]= $S^{i^*}{}_{j^*}\, x^{j^*}$

Out[21]= $S^{i}{}_{j}\, x^{j}$

```
Flavor::notflavor : blue is not a currently declared index flavor.
```

Out[22]= $S^{i}{}_{j}\, x^{j}$

Base indices can be changed in flavor only if they are integers.

```
In[23]:= step1
         % // ToFlavor[rocket]
         step2
         % // ToFlavor[Identity]
```

$$\text{Out[23]= } \{S^1{}_1\, x^1 + S^1{}_2\, x^2 + S^1{}_3\, x^3,\ S^2{}_1\, x^1 + S^2{}_2\, x^2 + S^2{}_3\, x^3,\ S^3{}_1\, x^1 + S^3{}_2\, x^2 + S^3{}_3\, x^3\}$$

$$\text{Out[24]= } \{S^{1^*}{}_{1^*}\, x^{1^*} + S^{1^*}{}_{2^*}\, x^{2^*} + S^{1^*}{}_{3^*}\, x^{3^*},\ S^{2^*}{}_{1^*}\, x^{1^*} + S^{2^*}{}_{2^*}\, x^{2^*} + S^{2^*}{}_{3^*}\, x^{3^*},\ S^{3^*}{}_{1^*}\, x^{1^*} + S^{3^*}{}_{2^*}\, x^{2^*} + S^{3^*}{}_{3^*}\, x^{3^*}\}$$

$$\text{Out[25]= } \{S^x{}_x\, x^x + S^x{}_y\, x^y + S^x{}_z\, x^z,\ S^y{}_x\, x^x + S^y{}_y\, x^y + S^y{}_z\, x^z,\ S^z{}_x\, x^x + S^z{}_y\, x^y + S^z{}_z\, x^z\}$$

$$\text{Out[26]= } \{S^x{}_x\, x^x + S^x{}_y\, x^y + S^x{}_z\, x^z,\ S^y{}_x\, x^x + S^y{}_y\, x^y + S^y{}_z\, x^z,\ S^z{}_x\, x^x + S^z{}_y\, x^y + S^z{}_z\, x^z\}$$

Applying the same flavor twice leaves the flavor unchanged.

```
In[27]:= CovariantD[Suu[i, j], k] // ToFlavor[red]
         % // ToFlavor[red, red]
         % // FullForm
```

$$\text{Out[27]= } S^{i\,j}{}_{;k}$$

$$\text{Out[28]= } S^{i\,j}{}_{;k}$$

```
Out[29]//FullForm=
CovariantD[Tensor[S, List[red[i], red[j]], List[Void, Void]], red[k]]
```

```
In[30]:= ClearTensorShortcuts[{x, 1}, {S, 2}]
```

This resets to the original state.

```
In[31]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor[IndexFlavors];
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

TotalD

- `TotalD[f, t]` gives the total derivative of f with respect to t.
- `TotalD[f, {t, v, ...}]` gives multiple derivatives.

Tensors are funtions of the coordinates and any variation of the Tensor is due to the variation of the coordiates over the parameter of differentiation used. Tensors are not allowed to depend on the parameters directly but only through the coordinates. See ExpandTotalD.

This derivative is ambiguous until we expand the partial derivative providing the specific coordinates using ExpandTotalD.

TotalD bears the same relation to AbsoluteD as PartialD does to CovariantD, i.e., it takes no notice of the metric.

Total derivative expressions are fully evaluated when a tensor is expanded to its components.

See also: `ExpandTotalD`, `CovariantD`, `PartialD`, `AbsoluteD`, `SetDerivativeSymbols`.

Examples

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

Save the Settings

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

In[6]:= DefineTensorShortcuts[{{x, S, T}, 1}, {{S, T}, 2}]
```

The total derivative works like a regular derivative, but allows the tensor formalism to be used. It is formatted on output but maintained with a TotalD header internally.

```
In[7]:= Su[i]
        TotalD[%, t]
        % // FullForm

Out[7]= S^i

Out[8]= dS^i/dt

Out[9]//FullForm=
TotalD[Tensor[S, List[i], List[Void]], t]
```

Higher order derivatives can be performed.

```
In[10]:= NestList[TotalD[#1, t] &, Su[m], 3]

Out[10]= {S^m, dS^m/dt, d^2 S^m/(dt dt), d^3 S^m/(dt dt dt)}
```

It can also be written...

```
In[11]:= Su[i]
         TotalD[%, {t, t, t}]
```

$$\text{Out[11]= } S^i$$

$$\text{Out[12]= } \frac{d^3 S^i}{dt\, dt\, dt}$$

Or you can differentiate with respect to different variables.

```
In[13]:= Su[i]
         TotalD[%, {t, u, v}]
```

$$\text{Out[13]= } S^i$$

$$\text{Out[14]= } \frac{d^3 S^i}{dt\, du\, dv}$$

Here are the first two derivatives of a tensor contraction. Nothing extra has to be done to accommodate flavored expressions.

```
In[15]:= Td[m] Tu[m] // ToFlavor[red]
         TotalD[%, {t, t}]
```

$$\text{Out[15]= } T_m\, T^m$$

$$\text{Out[16]= } T^m \frac{d^2 T_m}{dt\, dt} + 2 \frac{d T_m}{dt} \frac{d T^m}{dt} + T_m \frac{d^2 T^m}{dt\, dt}$$

With no arguments, we obtain a differential expression.

```
In[17]:= Td[m] Tu[m] // ToFlavor[red]
         TotalD[%]
```

$$\text{Out[17]= } T_m\, T^m$$

$$\text{Out[18]= } T^m\, dT_m + T_m\, dT^m$$

Here, we define the coordinate functions to be moving on a helix. We set the attributes of a and b to be Constant.

```
In[19]:= SetTensorValues[xu[i], {a Cos[t], a Sin[t], b t}, True]
         SetAttributes[#, Constant] & /@ {a, b};
```

Here is the vector expanded.

```
In[21]:= xu[i]
         % // EinsteinArray[]
```

$$\text{Out[21]= } x^i$$

$$\text{Out[22]= } \{a \text{Cos}[t], a \text{Sin}[t], b t\}$$

When we use `TotalD` and expand, we obtain the velocity vector.

```
In[23]:= xu[i]
         TotalD[%, t]
         % // EinsteinArray[]
```

$$\text{Out[23]=}\ x^i$$

$$\text{Out[24]=}\ \frac{\partial x^i}{\partial t}$$

```
Out[25]= {-a Sin[t], a Cos[t], b}
```

The second derivative gives the acceleration.

```
In[26]:= xu[i]
         TotalD[%, {t, t}]
         % // EinsteinArray[]
         ClearTensorValues[xu[i]]
```

$$\text{Out[26]=}\ x^i$$

$$\text{Out[27]=}\ \frac{\partial^2 x^i}{\partial t\, \partial t}$$

```
Out[28]= {-a Cos[t], -a Sin[t], 0}
```

`TotalD` will take derivatives of functions of components.

```
In[30]:= Sin[xu[1]]^2
         TotalD[%, {t, t}] // Expand
```

$$\text{Out[30]=}\ \text{Sin}[x^1]^2$$

$$\text{Out[31]=}\ 2\,\text{Cos}[x^1]^2\left(\frac{\partial x^1}{\partial t}\right)^2 - 2\,\text{Sin}[x^1]^2\left(\frac{\partial x^1}{\partial t}\right)^2 + 2\,\text{Cos}[x^1]\,\text{Sin}[x^1]\,\frac{\partial^2 x^1}{\partial t\, \partial t}$$

Restore settings.

```
In[32]:= ClearTensorValues /@ {xu[i]};
         ClearTensorShortcuts[{{x, S, T}, 1}, {{S, T}, 2}]
```

```
In[34]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         ClearAll[oldindices, oldflavors, metric, a, b]
```