SelectedTensorRules

- `SelectedTensorRules[`*label*, *pattern*`]` will select the rules for label whose right hand sides are nonzero and whose left hand sides match the pattern.

This is used to select and display independent, nonzero values of specific tensor forms

`SelectedTensorRules` does not simplify the right hand sides and checks only for explicit zero values. Values rules should be simplified as much as possible before they are stored and before `SelectedTensorRules` is used.

SelectedTensorRules should not be used for substitution as it may not contain the full set of values.

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

Examples

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

Save the settings.

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

We will set up the metric for a wormhole in general relativity and work in a red flavor. (From an excellent book *Gravity* by James B. Hartle.)

```
In[6]:= DeclareBaseIndices[{t, r, θ, ϕ}]
        DefineTensorShortcuts[{x, 1}, {g, 2}, {Γ, 3}]
        labs = {x, δ, g, Γ};
```

```
In[9]:= SetAttributes[b, Constant];
        (cmetric = DiagonalMatrix[{-1, 1, b^2 + r^2, (b^2 + r^2) Sin[θ]^2}]) // MatrixForm
        (metric = cmetric // CoordinatesToTensors[{t, r, θ, ϕ}, x, red]) // MatrixForm
        MapThread[SetTensorValueRules[#1, #2] &,
          {{gdd[a, b], guu[a, b]} // ToFlavor[red], {metric, Inverse@metric}}];
```

Out[10]//MatrixForm=

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & b^2+r^2 & 0 \\ 0 & 0 & 0 & (b^2+r^2)\,\mathrm{Sin}[\theta]^2 \end{pmatrix}$$

Out[11]//MatrixForm=

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & b^2+(x^{r})^2 & 0 \\ 0 & 0 & 0 & \mathrm{Sin}[x^{\theta}]^2\,(b^2+(x^{r})^2) \end{pmatrix}$$

Next we calculate and set rules for the Christoffel symbols.

```
In[13]:= MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]} // ToFlavor[red],
            CalculateChristoffels[labs, red, Simplify]}];
```

The following statement picks out just the independent nonzero up values for the Christoffel symbols, which are many fewer that the full set calculated.

```
In[14]:= SelectedTensorRules[Γ, Γudd[_, j_, k_] /; OrderedQ[{j, k}]] //
           UseCoordinates[{t, r, θ, ϕ}, x, red] // TableForm
```

Out[14]//TableForm=

$$\Gamma^{r}{}_{\theta\theta} \to -r$$
$$\Gamma^{r}{}_{\phi\phi} \to -r\,\mathrm{Sin}[\theta]^2$$
$$\Gamma^{\theta}{}_{r\theta} \to \frac{r}{b^2+r^2}$$
$$\Gamma^{\theta}{}_{\phi\phi} \to -\frac{1}{2}\,\mathrm{Sin}[2\,\theta]$$
$$\Gamma^{\phi}{}_{r\phi} \to \frac{r}{b^2+r^2}$$
$$\Gamma^{\phi}{}_{\theta\phi} \to \mathrm{Cot}[\theta]$$

Restore settings.

```
In[15]:= ClearTensorValues /@ ToFlavor[red] /@ {Γudd[i, j, k], Γddd[i, j, k]};
         ClearTensorShortcuts[{x, 1}, {g, 2}, {Γ, 3}]

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

SetCovariantDisplay

■ `SetCovariantDisplay[`*mode*`]` will set the display mode of unexpanded covariant derivatives where mode may be "SemicolonMode" or "DelMode" in quotes.

"SemiColonMode" will set the unevaluated covariant derivative display to a semicolon followed by the differentiation indices.

"DelMode" will set the unevaluated covariant derivative display to a ∇ with the differentiation indices as subscripts.

This only affects the formatted output, not the internal representation.

Tensorial is initialized with the "SemicolonMode".

SemicolonMode can use a symbol other than a semicolon by using SetDerivativeSymbols.

The argument must be a string

The formatted output may be copied and pasted.

See also: `OutputFormat`, `SetDerivativeSymbols`, `CovariantD`.

Examples

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

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

The normal display mode is SemicolonMode. It would be set as follows.

```
In[3]:= SetCovariantDisplay["SemicolonMode"]
```

In SemicolonMode the differentiation indices are slightly subscripted.

```
In[4]:= CovariantD[Sdd[a, b], {c, d}]
```

Out[4]= $S_{a\,b;c\,d}$

Here we use a vertical bar rather than a semicolon to distinguish the covariant derivative indices.

```
In[5]:= SetDerivativeSymbols[{",", "|", "d", "d", "D", "d", ""}];
        CovariantD[Sdd[a, b], {c, d}]
```

Out[6]= $S_{a\,b|c\,d}$

With a nested Tensor expression parentheses are used.

```
In[7]:= CovariantD[Tensor[Sdd[a, b] pu[c]], {d, e}]
```

Out[7]= $(p^{c}\,S_{a\,b})_{|d\,e}$

The following sets the DelMode.

```
In[8]:= SetCovariantDisplay["DelMode"]
```

```
In[9]:= CovariantD[Sdd[a, b], {c, d}]
```

Out[9]= $\nabla_{c\,d}S_{a\,b}$

```
In[10]:= CovariantD[Tensor[Sdd[a, b] pu[c]], {d, e}]
```

Out[10]= $\nabla_{d\,e}(p^{c}\,S_{a\,b})$

Reset the displays to default values.

```
In[11]:= SetCovariantDisplay["SemicolonMode"]
         SetDerivativeSymbols[{",", ";", "ⅆ", "ⅆ", "D", "d", ""}];
```

SetDerivativeSymbols

■ `SetDerivativeSymbols[{`*DifSym, CovSym, TDup, TDdown, ADup, ADdown, LDSym*`}]` sets the characters used in unevaluated derivative displays.

This only affects the formatted output, not the internal representation.

DifSym is the character used to prefix partial derivative indices. Default: ","

CovSym is the character used to prefix covariant derivative indices. Default: ";" (semi-colon)

TDup is the upper differentiation symbol in total derivatives. Default: "ⅆ"

TDdown is the lower differentiation symbol in total derivatives. Default: "ⅆ"

ADup is the upper differentiation symbol in absolute (intrinsic) derivatives. Default: "D"

ADdown is the lower differentiation symbol in absolute derivatives: Default: "d"

LDSym is the symbol for a Lie derivative: Default: "£" (Sterling symbol)

The items must be Strings, generally single character Strings.

LieD will remain unformatted unless SetLieDisplay is used.

See also: `SetLieDisplay`, `SetCovariantDisplay`, `TotalD`, `LieD`, `AbsoluteD`.

Examples

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

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

Tensorial has the following default derivative output formatting for differentiated indices, total derivatives and absolute derivatives.

```
In[3]:= expr1 = {PartialD[Su[i], j], PartialD[Su[i], {j, k}], CovariantD[Su[i], j],
           CovariantD[Su[i], {j, k}], TotalD[Su[i], t], TotalD[Su[i], {t, t}],
           AbsoluteD[Su[i], t], AbsoluteD[Su[i], {t, t}], LieD[Su[i], V]}
```

Out[3]= $\left\{S^{i}{}_{,j},\ S^{i}{}_{,jk},\ S^{i}{}_{;j},\ S^{i}{}_{;jk},\ \frac{ⅆS^{i}}{ⅆt},\ \frac{ⅆ^{2}S^{i}}{ⅆt\,ⅆt},\ \frac{D\,S^{i}}{d\,t},\ \frac{D^{2}S^{i}}{d\,t\,d\,t},\ \text{LieD}[S^{i}, V]\right\}$

These are some other possibilities. Some texts use | for a covariant derivative index.

```
In[4]:= SetDerivativeSymbols[{"∂", "|", "ⅆ", "ⅆ", "δ", "δ", "£"}];
        expr1
```

Out[5]= $\left\{S^{i}{}_{\partial j},\ S^{i}{}_{\partial jk},\ S^{i}{}_{|j},\ S^{i}{}_{|jk},\ \frac{ⅆS^{i}}{ⅆt},\ \frac{ⅆ^{2}S^{i}}{ⅆt\,ⅆt},\ \frac{\delta\,S^{i}}{\delta\,t},\ \frac{\delta^{2}S^{i}}{\delta\,t\,\delta\,t},\ \text{LieD}[S^{i}, V]\right\}$

The following sets "LieMode" for Lie derivatives and changes some of the other symbols.

```
In[6]:=  SetLieDisplay["LieMode"]
         SetDerivativeSymbols[{"∂", "𝔻", "d", "d", "∇", "d", "£"}];
         expr1
```

Out[8]= $\left\{S^{i}{}_{\partial j},\ S^{i}{}_{\partial j\,k},\ S^{i}{}_{\mathbb{D} j},\ S^{i}{}_{\mathbb{D} j\,k},\ \frac{d\,S^{i}}{d\,t},\ \frac{d^{2}\,S^{i}}{d\,t\,d\,t},\ \frac{\nabla S^{i}}{d\,t},\ \frac{\nabla^{2}\,S^{i}}{d\,t\,d\,t},\ \pounds_{V} S^{i}\right\}$

The following sets the "DelMode" for covariant derivatives and changes the Lie and some other symbols.

```
In[9]:=  SetCovariantDisplay["DelMode"]
         SetDerivativeSymbols[{",", ";", "d", "d", "𝔻", "ⅆ", "𝕃"}];
         expr1
```

Out[11]= $\left\{S^{i}{}_{,j},\ S^{i}{}_{,j\,k},\ \nabla_{j} S^{i},\ \nabla_{j\,k} S^{i},\ \frac{d\,S^{i}}{d\,t},\ \frac{d^{2}\,S^{i}}{d\,t\,d\,t},\ \frac{\mathbb{D} S^{i}}{ⅆt},\ \frac{\mathbb{D}^{2}\,S^{i}}{ⅆt\,ⅆt},\ \mathbb{L}_{V} S^{i}\right\}$

The following resets the derivative displays.

```
In[12]:=  SetCovariantDisplay["SemicolonMode"]
          SetLieDisplay["PlainMode"]
          SetDerivativeSymbols[{",", ";", "ⅆ", "ⅆ", "D", "d", "£"}];
          ClearTensorShortcuts[{{S}, 1}]
          Clear[expr1]
```

SetLieDisplay

- `SetLieDisplay[`*mode*`]` will set the display mode of unexpanded Lie derivatives where mode may be "PlainMode" or "LieMode" in quotes.

"PlainMode" gives default unformatted display.

"LieMode" will set the unevaluated Lie derivative display to a £ (Sterling symbol))with the differentiation fields as subscripts. The actual symbol used can be changed with SetDerivativeSymbols.

This only affects the formatted output, not the internal representation.

Tensorial is initialized with the "PlainMode". If PlainMode is set while already in PlainMode, warning messages will be given by *Mathematica* because there are no formatting definitions to clear.

The formatted output may be copied and pasted.

See also: `OutputFormat`, `SetDerivativeSymbols`, `CovariantD`.

Examples

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

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

Normally, the unexpanded LieD is unformatted.

```
In[3]:= LieD[pu[a], V]
```

Out[3]= LieD[p^a, V]

It can be formatted in common textbook style with...

```
In[4]:= SetLieDisplay["LieMode"]
        LieD[pu[a], V]
```

Out[5]= $£_V p^a$

Here we use a capital script L instead of the Sterling symbol.

```
In[6]:= SetDerivativeSymbols[{",", ";", "ⅆ", "ⅆ", "D", "d", "ℒ"}];
        LieD[pu[a], {U, V}]
```

Out[7]= $\mathcal{L}_{U\,V} p^a$

With a nested Tensor expression parentheses are used.

```
In[8]:= LieD[Tensor[pu[a]], V]
        % // UnnestTensor
```

Out[8]= $\mathcal{L}_V (p^a)$

Out[9]= $\mathcal{L}_V p^a$

Reset the displays to default values.

```
In[10]:= SetLieDisplay["PlainMode"]
         SetDerivativeSymbols[{",", ";", "ⅆ", "ⅆ", "D", "d", "£"}];

In[12]:= ClearTensorShortcuts[{{p}, 1}, {{S}, 2}]
```

SetPartialDisplay

- `SetPartialDisplay[`*mode*`]` will set the display mode of unexpanded partial derivatives where mode may be "CommaMode" or "PartialMode" in quotes.

"CommaMode" will set the unexpanded covariant derivative display to a comma followed by the differentiation indices.

"PartialMode" will set the unexpanded partial derivative display to a ∂ with the differentiation indices as subscripts.

This only affects the formatted output, not the internal representation.

Tensorial is initialized with the "CommaMode".

CommaMode can use a symbol other than a comma by using SetDerivativeSymbols.

The argument must be a string

The formatted output may be copied and pasted.

See also: `OutputFormat`, `SetDerivativeSymbols`, `PartialD`.

Examples

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

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

The normal display mode for unexpanded partial derivatives is CommaMode. It would be set as follows.

```
In[3]:= SetPartialDisplay["CommaMode"]
```

In CommaMode the differentiation indices are slightly subscripted.

```
In[4]:= PartialD[Sdd[a, b], {c, d}]
```

Out[4]= $S_{a\,b,c\,d}$

Here we use a backslash rather than a comma to distinguish the partial derivative indices.

```
In[5]:= SetDerivativeSymbols[{"\\", ";", "ⅆ", "ⅆ", "D", "d", ""}];
        PartialD[Sdd[a, b], {c, d}]
```

Out[6]= $S_{a\,b\backslash c\,d}$

With a nested Tensor expression, or PartialD with a HoldOp, parentheses are used.

```
In[7]:= PartialD[Tensor[Sdd[a, b] pu[c]], {d, e}]
```

Out[7]= $(p^{c}\, S_{a\,b})_{\backslash d\,e}$

```
In[8]:= PartialD[Sdd[a, b] pu[c], {d, e}] // HoldOp[PartialD]
        % // ReleaseHold
```

Out[8]= $(p^{c} S_{a b})_{\backslash d e}$

Out[9]= $p^{c}{}_{\backslash e} S_{a b\backslash d} + p^{c}{}_{\backslash d} S_{a b\backslash e} + S_{a b\backslash e d} p^{c} + p^{c}{}_{\backslash e d} S_{a b}$

The following sets the PartialMode.

```
In[10]:= SetPartialDisplay["PartialMode"]
```

```
In[11]:= PartialD[Sdd[a, b], {c, d}]
```

Out[11]= $\partial_{c d} S_{a b}$

```
In[12]:= PartialD[Sdd[a, b] pu[c], {d, e}] // HoldOp[PartialD]
         % // ReleaseHold
```

Out[12]= $\partial_{d e} (p^{c} S_{a b})$

Out[13]= $\partial_{e} p^{c} \partial_{d} S_{a b} + \partial_{d} p^{c} \partial_{e} S_{a b} + \partial_{e d} S_{a b} p^{c} + \partial_{e d} p^{c} S_{a b}$

Reset the displays to default values.

```
In[14]:= SetPartialDisplay["CommaMode"]
         SetDerivativeSymbols[{",", ";", "d", "d", "D", "d", ""}];
```

SetRicciContraction

■ `SetRicciContraction[`*index*`]` will set the index of the Riemann tensor that will be contracted to form the Ricci tensor.

The contraction will occur on the first (up) index of the Riemann tensor and the specified index, which must be 3 or 4.

The default index is 3.

See also: `CalculateRiemannd`, `CalculateRRRG`.

Example

See the example in `CalculateRRRG`.

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

SetScalarSingleCovariantD

■ SetScalarSingle`CovariantD[`*on* : *True*`]` will enable or disable the coversion of a single covariant derivative of a scalar tensor to a partial derivative.

The optional argument may be True or False and the default is True.

Upon initialization of Tensorial the feature is enabled

See also: `CovariantD`, `ExpandCovariantD`, `PartialD`.

Examples

```
In[2]:= Needs["TensorCalculus4`Tensorial`"]
```

A single covariant differentiation of a scalar tensor will normally be converted to a partial differentiation.

```
In[3]:= Tensor[ϕ]
        CovariantD[%, i]
```

Out[3]= ϕ

Out[4]= $\phi_{,i}$

Multiple covariant differentiations done at once will not result in a partial derivative.

```
In[5]:= Tensor[ϕ]
        CovariantD[%, {i, j}]
        % // ExpandCovariantD[{x, δ, g, Γ}, {a, b}]
```

Out[5]= ϕ

Out[6]= $\phi_{;i\,j}$

Out[7]= $\frac{\partial^2 \phi}{\partial x^j\, \partial x^i} - \Gamma^{b}{}_{j\,i}\, \frac{\partial \phi}{\partial x^b}$

However, if the covariant differentiations are done in steps it may result in an undesired expression.

```
In[8]:= Tensor[ϕ]
        CovariantD[%, i]
        CovariantD[%, j]
```

Out[8]= ϕ

Out[9]= $\phi_{,i}$

Out[10]= $(\phi_{,i})_{;j}$

To avoid these situations the feature that turns single covariant derivatives of scalar tensors into partial derivatives may be disabled.

```
In[11]:= SetScalarSingleCovariantD[False]

In[12]:= Tensor[ϕ]
         CovariantD[%, i]
         CovariantD[%, j]
```

Out[12]= ϕ

Out[13]= $\phi_{;i}$

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

On expansion with scalar tensors the first differentiation becomes a partial differentiation.

```
In[15]:= Tensor[ϕ]
         CovariantD[%, i]
         % // ExpandCovariantD[{x, δ, g, Γ}, a]
```

Out[15]= ϕ

Out[16]= $\phi_{;i}$

Out[17]= $\frac{\partial \phi}{\partial x^{i}}$

```
In[18]:= SetScalarSingleCovariantD[True]
```

SetTensorValueRules

- `SetTensorValueRules[`*tensortemplate, values, permissive : False*`]` will add substitution `TensorValueRules[label]`.
- `SetTensorValueRules[`*Tensor*[*label*], *value*`]` will create substitution rules for a scalar te

tensortemplate can be a tensor shortcut, e.g., `Tdd[a, b]`, or a FullForm tensor express `Tensor[label, upindices, downindices]`. There are no shortcuts for scalar

The order of components in values, that is the levels in the tensor array, will correspond template, not necessarily their slot order in the tensor. It will be good practice to usually as the slot order, unless you are deliberately using transposes.

The dimension of each index is that associated with the index flavor by the `DeclareBa`

The number of Dimensions of the values array must be greater than or equal to the numb this does allow the setting of arrays as values of tensor components.

When arrays are given as component values then all dimensions of the array must be equ permissive to True waives this rule.

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

See also: `SetTensorValues`, `ClearTensorValues`, `Tensor`, `UseCoordinat`

Examples

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

Save the settings and declare base indices and flavors.

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

We define some tensor shortcuts.

```
In[7]:= DefineTensorShortcuts[{{v, g, e}, 1}, {h, 2}]
```

Here is a tensor. It has no values.

```
In[8]:= vu[i]
        % // EinsteinArray[]
```

Out[8]= v^i

Out[9]= $\{v^1, v^2, v^3\}$

Now we create substitution rules for the values of the contravariant components of `v`. When the without their values. But the values can be substituted with `TensorValueRules[v]`

```
In[10]:= SetTensorValueRules[vu[i], {1, 2, 3}]
         vu[i]
         % // EinsteinArray[]
         % /. TensorValueRules[v]
```

Out[11]= v^i

Out[12]= $\{v^1, v^2, v^3\}$

Out[13]= $\{1, 2, 3\}$

Here, we look at the explicit rules. Only the contravariant components have values.

```
In[14]:= TensorValueRules[v]
```

Out[14]= $\{v^1 \to 1, v^2 \to 2, v^3 \to 3\}$

We can erase the values using ...

```
In[15]:= ClearTensorValues[vu[i]];
         TensorValueRules[v]
```

Out[16]= $\{\}$

Here is an order 2 tensor, `h`.

```
In[17]:= hud[i, j]
         % // EinsteinArray[] // MatrixForm
```

Out[17]= ${h^i}_j$

Out[18]//MatrixForm=

$$\begin{pmatrix} {h^1}_1 & {h^1}_2 & {h^1}_3 \\ {h^2}_1 & {h^2}_2 & {h^2}_3 \\ {h^3}_1 & {h^3}_2 & {h^3}_3 \end{pmatrix}$$

We will create substitution rules for the components of `h`.

```
In[19]:= (hvalues = Table[j - i, {i, 1, 3}, {j, 1, 3}]) // MatrixForm
```

Out[19]//MatrixForm=

$$\begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

```
In[20]:= SetTensorValueRules[hud[i, j], hvalues]
         TensorValueRules[h]
         hud[i, j]
         % // EinsteinArray[] // MatrixForm
         % /. TensorValueRules[h] // MatrixForm
```

Out[21]= $\{h^1{}_1 \to 0,\ h^1{}_2 \to 1,\ h^1{}_3 \to 2,\ h^2{}_1 \to -1,\ h^2{}_2 \to 0,\ h^2{}_3 \to 1,\ h^3{}_1 \to -2,\ h$

Out[22]= $h^i{}_j$

Out[23]//MatrixForm=

$$\begin{pmatrix} h^1{}_1 & h^1{}_2 & h^1{}_3 \\ h^2{}_1 & h^2{}_2 & h^2{}_3 \\ h^3{}_1 & h^3{}_2 & h^3{}_3 \end{pmatrix}$$

Out[24]//MatrixForm=

$$\begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

If we reverse the indices in the tensor template when using SetTensorValueRules, then we obtai
you deliberately want to put in the transposed values. Normally you should put the indices into t

```
In[25]:= SetTensorValueRules[hud[j, i], hvalues]
         TensorValueRules[h]
         hud[i, j]
         % // EinsteinArray[] // MatrixForm
         % /. TensorValueRules[h] // MatrixForm
```

Out[26]= $\{h^1{}_1 \to 0,\ h^2{}_1 \to 1,\ h^3{}_1 \to 2,\ h^1{}_2 \to -1,\ h^2{}_2 \to 0,\ h^3{}_2 \to 1,\ h^1{}_3 \to -2,\ h$

Out[27]= $h^i{}_j$

Out[28]//MatrixForm=

$$\begin{pmatrix} h^1{}_1 & h^1{}_2 & h^1{}_3 \\ h^2{}_1 & h^2{}_2 & h^2{}_3 \\ h^3{}_1 & h^3{}_2 & h^3{}_3 \end{pmatrix}$$

Out[29]//MatrixForm=

$$\begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}$$

```
In[30]:= ClearTensorValues[hud[i, j]]
         TensorValueRules[h]

Out[31]= {}
```

We can also create rules that substitute arrays for tensor components. The last index could be as
Let g_i be a set of basis vectors.

```
In[32]:= SetTensorValueRules[gd[red@i], ( 1  0  3
                                          0  2  1
                                          2  2  3 )]

         TensorValueRules[g]

Out[33]= {g1 → {1, 0, 3}, g2 → {0, 2, 1}, g3 → {2, 2, 3}}
```

```
In[34]:= ClearTensorValues[gd[red@i]]
         TensorValueRules[g]

Out[35]= {}
```

Suppose we want to set tangent vectors in the 3D embedding space of the following 2D manifol

```
In[36]:= ξ[u_, v_] := {u + v, u - v, 2 u v}
```

If we try to directly set 3D tensor values they don't set because the dimensions must match the d

```
In[37]:= DeclareBaseIndices[{u, v}]
         SetTensorValueRules[ed[i], {∂u ξ[u, v], ∂v ξ[u, v]}]

         SetTensorValueRules::size :
          The size of each value array must be the same as the dimension of the

Out[38]= $Failed
```

```
In[39]:= TensorValueRules[e]

Out[39]= TensorValueRules[e]
```

By setting the optional argument permissive to True the values can be set.

```
In[40]:= SetTensorValueRules[ed[i], {∂u ξ[u, v], ∂v ξ[u, v]}, True]
```

```
In[41]:= TensorValueRules[e]

Out[41]= {eu → {1, 1, 2 v}, ev → {1, -1, 2 u}}
```

```
In[42]:= ClearTensorValues[ed[i]]
```

We can create and store values for rectangular arrays by using different flavor indices and associ

```
In[43]:= DeclareBaseIndices[{1, 2}, {blue, {1, 2, 3}}]
```

```
In[44]:= SetTensorValueRules[hdd[i, blue@j], (1 0 2
                                              0 1 3), True]
         TensorValueRules[h]
         ClearTensorValues[hdd[i, blue@j]]
```

Out[45]= $\{h_{1\,1} \to 1,\ h_{1\,2} \to 0,\ h_{1\,3} \to 2,\ h_{2\,1} \to 0,\ h_{2\,2} \to 1,\ h_{2\,3} \to 3\}$

```
In[47]:= DeclareBaseIndices[{1, 2, 3}]
```

We can set values for only selected components of a tensor by creating a corresponding array an
rules for the set values are created.

```
In[48]:= mat = hdd[i, j] // ToArrayValues[]
         Part[mat, {1, 2}, {1, 2}] = (2 0
                                      1 3);
         mat // MatrixForm
         SetTensorValueRules[hdd[i, j], mat]
         TensorValueRules[h]
```

Out[48]= $\{\{h_{1\,1},\ h_{1\,2},\ h_{1\,3}\},\ \{h_{2\,1},\ h_{2\,2},\ h_{2\,3}\},\ \{h_{3\,1},\ h_{3\,2},\ h_{3\,3}\}\}$

Out[50]//MatrixForm=

$$\begin{pmatrix} 2 & 0 & h_{1\,3} \\ 1 & 3 & h_{2\,3} \\ h_{3\,1} & h_{3\,2} & h_{3\,3} \end{pmatrix}$$

Out[52]= $\{h_{1\,1} \to 2,\ h_{1\,2} \to 0,\ h_{2\,1} \to 1,\ h_{2\,2} \to 3\}$

Rules can also be created for scalar tensors. (Note that `Tensor[ϕ]` displays as just ϕ.)

```
In[53]:= Clear[ϕ];
         SetTensorValueRules[Tensor[ϕ], t^2]
         TensorValueRules[ϕ]
```

Out[55]= $\{\phi \to t^2\}$

```
In[56]:= Tensor[ϕ]
         % /. TensorValueRules[ϕ]
```

Out[56]= ϕ

Out[57]= t^2

The rules can be cleared with...

```
In[58]:= ClearTensorValues[Tensor[ϕ]]
         TensorValueRules[ϕ]
```

Out[59]= { }

It is permissible to have the same symbol for the tensor label, a pattern index, and as one of the v

```
In[60]:= SetTensorValueRules[vu[v], {v, 1, 2}]
         TensorValueRules[v]
```

Out[61]= $\{v^1 \to v, v^2 \to 1, v^3 \to 2\}$

```
In[62]:= ClearTensorShortcuts[{{v, g}, 1}, {h, 2}]
```

Restore the original state...

```
In[63]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors, ξ, mat]
```

```
In[67]:= ClearTensorShortcuts[{{v, g, e}, 1}, {h, 2}]
```

SetTensorValues

- `SetTensorValues[`*tensortemplate, values, permissive : False*`]` will set the component values of the tensor as UpValues for label.
- `SetTensorValues[`*Tensor*[*label*], *value*`]` set the value for a scalar tensor.

tensortemplate can be a tensor shortcut, e.g., `Tdd[a, b]`, or a FullForm tensor expression, `Tensor[label, upindices, downindices]`. There are no shortcuts for scalar tensors.

The order of components in values, that is the levels in the tensor array, will correspond to the Sort order of the raw indices in the tensor template, not necessarily their slot order in the tensor. It will be good practice to usually make the sort order of the template indices the same as the slot order, unless you are deliberately using transposes.

The dimension of each index is that associated with the index flavor by the `DeclareBaseIndices` statement.

The number of Dimensions of the values tensor must be greater than or equal to the number of indices. Although they will normally be equal, this does allow the setting of arrays as values of tensor components.

When arrays are given as component values then all dimensions of the array must be equal to the dimension of the associated index. Setting permissive to True waives this rule.

Shortcuts can be used as the tensor pattern, but there are no shortcuts for scalar tensors.

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

Examples

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

Save the settings and declare base indices and flavors.

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

We define some tensor shortcuts.

```
In[7]:= DefineTensorShortcuts[{{v, g, e}, 1}, {h, 2}]
```

Here is a tensor. It has no values.

```
In[8]:= vu[i]
        % // EinsteinArray[]
```

Out[8]= v^i

Out[9]= $\{v^1, v^2, v^3\}$

Now we create values for the contravariant components of `v`. When the tensor is expanded it takes on the values.

```
In[10]:= SetTensorValues[vu[i], {1, 2, 3}]
         vu[i]
         % // EinsteinArray[]
```

Out[11]= v^{i}

```
Out[12]= {1, 2, 3}
```

The definitions are stored in...

```
In[13]:= UpValues[v]
```

Out[13]= {HoldPattern[v^{1}] :→ 1, HoldPattern[v^{2}] :→ 2, HoldPattern[v^{3}] :→ 3}

Only the contravariant components have values.

We can erase the values using ...

```
In[14]:= ClearTensorValues[vu[i]];
         UpValues[v]

Out[15]= {}
```

Here is an order 2 tensor, h.

```
In[16]:= hud[i, j]
         % // EinsteinArray[] // MatrixForm
```

Out[16]= ${h^{i}}_{j}$

Out[17]//MatrixForm=

$$\begin{pmatrix} {h^{1}}_{1} & {h^{1}}_{2} & {h^{1}}_{3} \\ {h^{2}}_{1} & {h^{2}}_{2} & {h^{2}}_{3} \\ {h^{3}}_{1} & {h^{3}}_{2} & {h^{3}}_{3} \end{pmatrix}$$

We will set the following values for the components of h.

```
In[18]:= (hvalues = Table[j - i, {i, 1, 3}, {j, 1, 3}]) // MatrixForm
```

Out[18]//MatrixForm=

$$\begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

```
In[19]:= SetTensorValues[hud[i, j], hvalues]
         hud[i, j]
         % // EinsteinArray[] // MatrixForm
```

Out[20]= ${h^{i}}_{j}$

Out[21]//MatrixForm=

$$\begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

If we reverse the indices in the tensor template when using SetTensorValues, then we obtain the transposed values. You should only do this if you deliberately want to put in the transposed values. Normally you should put the indices into the slots in sort order.

```
In[22]:=  SetTensorValues[hud[j, i], hvalues]
          hud[i, j]
          % // EinsteinArray[] // MatrixForm
```

Out[23]= $h^{i}{}_{j}$

Out[24]//MatrixForm=

$$\begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix}$$

```
In[25]:=  ClearTensorValues[hud[i, j]]
```

We can also create rules that substitute arrays for tensor components. The last index could be associated with arrays with any number of levels. Let g_i be a set of basis vectors.

In[26]:= SetTensorValues$\left[\text{gd[red@i]}, \begin{pmatrix} 1 & 0 & 3 \\ 0 & 2 & 1 \\ 2 & 2 & 3 \end{pmatrix}\right]$

```
          gd[red@i]
          % // EinsteinArray[]
```

Out[27]= g_i

```
Out[28]=  {{1, 0, 3}, {0, 2, 1}, {2, 2, 3}}

In[29]:=  ClearTensorValues[gd[red@i]]
```

Suppose we want to set tangent vectors in the 3D embedding space of the following 2D manifold.

```
In[30]:=  ξ[u_, v_] := {u + v, u - v, 2 u v}
```

If we try to directly set 3D tensor values they don't set because the dimensions must match the dimension associated with the index.

```
In[31]:=  DeclareBaseIndices[{u, v}]
          SetTensorValues[ed[i], {∂u ξ[u, v], ∂v ξ[u, v]}]

        SetTensorValues::size : The size of each value array
          must be the same as as the dimension of the corresponding index.

Out[32]=  $Failed
```

By setting the optional argument permissive to True the values can be set.

```
In[33]:=  SetTensorValues[ed[i], {∂u ξ[u, v], ∂v ξ[u, v]}, True]
          ed[i]
          % // EinsteinArray[]
```

Out[34]= e_i

```
Out[35]=  {{1, 1, 2 v}, {1, -1, 2 u}}

In[36]:=  ClearTensorValues[ed[i]]
```

We can create and store values for rectangular arrays by using different flavor indices and associating different base indices with each flavor.

```
In[37]:=  DeclareBaseIndices[{1, 2}, {blue, {1, 2, 3}}]
```

```
In[38]:= SetTensorValues[hdd[i, blue@j], (1 0 2
                                          0 1 3), True]
         hdd[i, blue@j]
         % // EinsteinArray[] // MatrixForm
         ClearTensorValues[hdd[i, blue@j]]
```

Out[39]= $h_{i\,j}$

Out[40]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \end{pmatrix}$$

```
In[42]:= DeclareBaseIndices[{1, 2, 3}]
```

We can set values for only selected components of a tensor by creating a corresponding array and then using it to set the values. Notice that only rules for the set values are created.

```
In[43]:= mat = hdd[i, j] // ToArrayValues[];
         Part[mat, {1, 2}, {1, 2}] = (2 0
                                      1 3);
         mat // MatrixForm
         SetTensorValues[hdd[i, j], mat]
         hdd[i, j]
         % // EinsteinArray[] // MatrixForm
```

Out[45]//MatrixForm=

$$\begin{pmatrix} 2 & 0 & h_{1\,3} \\ 1 & 3 & h_{2\,3} \\ h_{3\,1} & h_{3\,2} & h_{3\,3} \end{pmatrix}$$

Out[47]= $h_{i\,j}$

Out[48]//MatrixForm=

$$\begin{pmatrix} 2 & 0 & h_{1\,3} \\ 1 & 3 & h_{2\,3} \\ h_{3\,1} & h_{3\,2} & h_{3\,3} \end{pmatrix}$$

Definitions can also be created for scalar tensors. (Note that `Tensor[ϕ]` displays as just ϕ.)

```
In[49]:= Clear[ϕ];
         SetTensorValues[Tensor[ϕ], t^2]
         UpValues[ϕ]
         Tensor[ϕ]
```

Out[51]= {HoldPattern[ϕ] :→ t^2}

Out[52]= t^2

The definition can be cleared with...

```
In[53]:= ClearTensorValues[Tensor[ϕ]]
         UpValues[ϕ]
```

```
Out[54]= {}
```

It is permissible to have the same symbol for the tensor label, a pattern index, and as one of the values.

```
In[55]:= SetTensorValues[vu[v], {v, 1, 2}]
         vu[v]
         % // EinsteinArray[]
```

Out[56]= v^v

```
Out[57]= {v, 1, 2}
```

```
In[58]:= ClearTensorShortcuts[{{v, g}, 1}, {h, 2}]
```

Restore the original state...

```
In[59]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors, ξ, mat]
```

```
In[63]:= ClearTensorShortcuts[{{v, g, e}, 1}, {h, 2}]
```

SimplifyTensorSum

■ `SimplifyTensorSum[`*expr*`]` will check that all terms in a tensor sum have valid indices, that the free indices are the same in all terms, and will simplify the sum by matching dummy indices in all terms that have the same index structure.

Terms which have the same tensor labels and index structure will be made the same, but subparts of a term that are similar to other terms may not be the same.

Free indices are indices that appear only once in each term. They must be the same in all terms. Dummy indices are indices that appear exactly once in an up position and once in a down position. All other indices are bad.

If the free indices do not match or there are bad indices in any terms, error messages are given and the simplification is aborted.

IndexChange can be used to perform specific reindexing under the user's control.

TensorSimplify automatically applies SimplifyTensorSum to terms with similar patterns, takes into account declared symmetries and is the routine of choice.

See also: `TensorSimplify`, `PatternReplaceIndex`, `UpDownAdjust`, `UpDownSwap`, `IndexChange`, `SymmetrizeSlots`, `SymmetrizePattern`, `NondependentPartialD`, `MapLevelParts`, `MapLevelPatterns`, `DeclareTensorSymmetries`, `DeclarePatternSymmetries`.

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[{{b, c, A, B, G}, 1}, {A, 2}, {{a, b, F}, 3}, {F, 4}]
```

Terms with the same labels and index structure are made the same.

```
In[6]:= c Ad[i] Bu[i] + a Ad[j] Bu[j] + b Ad[m] Bu[m]
        % // SimplifyTensorSum
        % // Simplify
```

Out[6]= $c A_i B^i + a A_j B^j + b A_m B^m$

Out[7]= $a A_i B^i + b A_i B^i + c A_i B^i$

Out[8]= $(a + b + c) A_i B^i$

```
In[9]:=  c Ad[i] Bu[i] + a Ad[j] Bu[j] + b Ad[m] Bu[m] // ToFlavor[red]
         % // SimplifyTensorSum
         % // Simplify
```

Out[9]= $c A_i B^i + a A_j B^j + b A_m B^m$

Out[10]= $a A_i B^i + b A_i B^i + c A_i B^i$

Out[11]= $(a + b + c) A_i B^i$

This sum is bad because the free indices do not match.

```
In[12]:=  a Auu[i, j] Gd[i] + b Auu[k, m] Gd[k]
          % // SimplifyTensorSum
```

Out[12]= $a A^{ij} G_i + b A^{km} G_k$

```
SimplifyTensorSum::free : Free indices are not the same in all terms.
```

Out[13]= $Aborted

The second term of this sum contains a bad index. k is repeated 3 times.

```
In[14]:=  auud[i, k, j] budd[j, k, m] + 2 auud[i, j, k] budd[k, j, m] cu[k] // ToFlavor[red]
          % // SimplifyTensorSum
```

Out[14]= $a^{ik}{}_j b^j{}_{km} + 2 a^{ij}{}_k b^k{}_{jm} c^k$

SimplifyTensorSum::badterm : Term $2 a^{ij}{}_k b^k{}_{jm} c^k$ contains bad indices.

Out[15]= $Aborted

```
In[16]:=  a Fddu[i, j, i] + b Fddu[k, j, k] + c Fddu[m, j, m]
          % // SimplifyTensorSum // Simplify
```

Out[16]= $a F_{ij}{}^i + b F_{kj}{}^k + c F_{mj}{}^m$

Out[17]= $(a + b + c) F_{ij}{}^i$

TensorSimplify, which uses SimplifyTensorSum, combines only the first two terms in this expression. We can use IndexChange to obtain a full simplification.

```
In[18]:=  a Fdduu[i, j, i, j] + b Fdduu[j, p, j, p] + c Ad[i] Bu[i] Fdduu[m, r, m, r]
          % // TensorSimplify
          MapAt[IndexChange[{{i, j}, {j, p}}], %, 1] // Simplify
```

Out[18]= $a F_{ij}{}^{ij} + b F_{jp}{}^{jp} + c A_i B^i F_{mr}{}^{mr}$

Out[19]= $(a + b) F_{ij}{}^{ij} + c A_i B^i F_{jp}{}^{jp}$

Out[20]= $(a + b + c A_i B^i) F_{jp}{}^{jp}$

This can be done more easily with PatternReplaceIndex

```
In[21]:= a Fdduu[i, j, i, j] + b Fdduu[j, p, j, p] + c Ad[i] Bu[i] Fdduu[m, r, m, r]
         % // PatternReplaceIndex[{j, p}, Fdduu[a_, b_, a_, b_]] // Simplify
```

Out[21]= $a\,F_{ij}{}^{ij} + b\,F_{jp}{}^{jp} + c\,A_i\,B^i\,F_{mr}{}^{mr}$

Out[22]= $(a + b + c\,A_i\,B^i)\,F_{jp}{}^{jp}$

```
In[23]:= (addd[r, s, t] + addd[s, r, t] + addd[s, t, r]) bu[r] bu[s] bu[t]
         % // SimplifyTensorSum
```

Out[23]= $(a_{rst} + a_{srt} + a_{str})\,b^r\,b^s\,b^t$

Out[24]= $3\,a_{rst}\,b^r\,b^s\,b^t$

Here, the absolute derivative of a tensor product is taken. It expands to 14 terms. `SimplifyTensorSum` reduces it to 12 terms. Total derivatives of the Christoffel symbol were set to zero. The other terms do not combine because the dummy indices are not in the same place.

```
In[25]:= Tensor[G, {Void}, {k}] Tensor[F, {m}, {Void}]
         expr1 =
          (AbsoluteD[%, {t, t}] // ExpandAbsoluteD[{x, δ, g, Γ}, {{a, b}, {c, d}}] // Expand) /.
           TotalD[Tensor[Γ, _, _], _] :→ 0
         Length[expr1]
         expr2 = expr1 // SimplifyTensorSum
         Length[%]
```

Out[25]= $F^m\,G_k$

Out[26]=
$$G_k \frac{\mathrm{d}^2 F^m}{\mathrm{d}t\,\mathrm{d}t} + 2\frac{\mathrm{d}F^m}{\mathrm{d}t}\frac{\mathrm{d}G_k}{\mathrm{d}t} + F^m \frac{\mathrm{d}^2 G_k}{\mathrm{d}t\,\mathrm{d}t} + G_k\,\Gamma^m{}_{ab}\frac{\mathrm{d}F^a}{\mathrm{d}t}\frac{\mathrm{d}x^b}{\mathrm{d}t} -$$
$$F^m\,\Gamma^a{}_{bk}\frac{\mathrm{d}G_a}{\mathrm{d}t}\frac{\mathrm{d}x^b}{\mathrm{d}t} + 2\,F^a\,\Gamma^m{}_{ab}\frac{\mathrm{d}G_k}{\mathrm{d}t}\frac{\mathrm{d}x^b}{\mathrm{d}t} - F^m\,G_a\,\Gamma^a{}_{bk}\frac{\mathrm{d}^2 x^b}{\mathrm{d}t\,\mathrm{d}t} + F^a\,G_k\,\Gamma^m{}_{ab}\frac{\mathrm{d}^2 x^b}{\mathrm{d}t\,\mathrm{d}t} +$$
$$G_k\,\Gamma^m{}_{cd}\frac{\mathrm{d}F^c}{\mathrm{d}t}\frac{\mathrm{d}x^d}{\mathrm{d}t} - 2\,G_c\,\Gamma^c{}_{dk}\frac{\mathrm{d}F^m}{\mathrm{d}t}\frac{\mathrm{d}x^d}{\mathrm{d}t} - F^m\,\Gamma^c{}_{dk}\frac{\mathrm{d}G_c}{\mathrm{d}t}\frac{\mathrm{d}x^d}{\mathrm{d}t} +$$
$$F^m\,G_a\,\Gamma^a{}_{bc}\,\Gamma^c{}_{dk}\frac{\mathrm{d}x^b}{\mathrm{d}t}\frac{\mathrm{d}x^d}{\mathrm{d}t} - 2\,F^a\,G_c\,\Gamma^c{}_{dk}\,\Gamma^m{}_{ab}\frac{\mathrm{d}x^b}{\mathrm{d}t}\frac{\mathrm{d}x^d}{\mathrm{d}t} + F^a\,G_k\,\Gamma^c{}_{ab}\,\Gamma^m{}_{cd}\frac{\mathrm{d}x^b}{\mathrm{d}t}\frac{\mathrm{d}x^d}{\mathrm{d}t}$$

Out[27]= 14

Out[28]=
$$G_k \frac{\mathrm{d}^2 F^m}{\mathrm{d}t\,\mathrm{d}t} + 2\frac{\mathrm{d}F^m}{\mathrm{d}t}\frac{\mathrm{d}G_k}{\mathrm{d}t} + F^m \frac{\mathrm{d}^2 G_k}{\mathrm{d}t\,\mathrm{d}t} + 2\,G_k\,\Gamma^m{}_{ab}\frac{\mathrm{d}F^a}{\mathrm{d}t}\frac{\mathrm{d}x^b}{\mathrm{d}t} - 2\,G_a\,\Gamma^a{}_{bk}\frac{\mathrm{d}F^m}{\mathrm{d}t}\frac{\mathrm{d}x^b}{\mathrm{d}t} -$$
$$2\,F^m\,\Gamma^a{}_{bk}\frac{\mathrm{d}G_a}{\mathrm{d}t}\frac{\mathrm{d}x^b}{\mathrm{d}t} + 2\,F^a\,\Gamma^m{}_{ab}\frac{\mathrm{d}G_k}{\mathrm{d}t}\frac{\mathrm{d}x^b}{\mathrm{d}t} - F^m\,G_a\,\Gamma^a{}_{bk}\frac{\mathrm{d}^2 x^b}{\mathrm{d}t\,\mathrm{d}t} + F^a\,G_k\,\Gamma^m{}_{ab}\frac{\mathrm{d}^2 x^b}{\mathrm{d}t\,\mathrm{d}t} +$$
$$F^m\,G_a\,\Gamma^a{}_{bd}\,\Gamma^d{}_{ck}\frac{\mathrm{d}x^b}{\mathrm{d}t}\frac{\mathrm{d}x^c}{\mathrm{d}t} + F^a\,G_k\,\Gamma^d{}_{ab}\,\Gamma^m{}_{dc}\frac{\mathrm{d}x^b}{\mathrm{d}t}\frac{\mathrm{d}x^c}{\mathrm{d}t} - 2\,F^a\,G_b\,\Gamma^b{}_{dk}\,\Gamma^m{}_{ac}\frac{\mathrm{d}x^c}{\mathrm{d}t}\frac{\mathrm{d}x^d}{\mathrm{d}t}$$

Out[29]= 12

Restore old settings...

```
In[30]:= ClearTensorShortcuts[{{b, c, A, B, G}, 1}, {A, 2}, {{a, b, F}, 3}, {F, 4}]
```

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

SumExpansion

- `SumExpansion[`*i, j, ..., base : Automatic*`][expr]` will sum on the indices i, j,..., in the expression. The range of the sum is over the base list, which has the default value of Automatic.
- `SumExpansion[{`*i, j, ...*`}, `*base : Automatic*`][expr]` may also be used.

The summation indices must carry their flavors. base indices are not flavored.

The expansion will be done on individual terms, terms in sums, on both sides of an equation, and over arrays.

Otherwise SumExpansion expands the entire expression it is applied to. Sometimes you may wish to Map it to more specific subexpressions.

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 optional argument base gives the base indices over which each index is summed. The default value is `Automatic` and then each index is summed over the complete set of base indices for the corresponding flavor. If a list of subsets of selected base indices is given then each index is summed over the corresponding selected subset taken in corresponding order. If a single list of selected base indices is supplied, then it will apply only to the first index.

`EinsteinSum`, which automatically expands on repeated up/down indices, will usually be more convenient to use.

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

Examples

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

Save the settings and declare base indices and flavors.

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

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

```
In[8]:= xu[i] yd[i]
        % // SumExpansion[i]
```

Out[8]= $x^i\, y_i$

Out[9]= $x^1\, y_1 + x^2\, y_2 + x^3\, y_3$

SumExpansion does not check for proper index notation.

```
In[10]:= xu[i] yu[i]
         % // SumExpansion[i]
```

Out[10]= $x^i\ y^i$

Out[11]= $x^1\ y^1 + x^2\ y^2 + x^3\ y^3$

The expansion will not occur unless the summation index carries the matching flavor.

```
In[12]:= xu[i] yd[i] // ToFlavor[red]
         % // SumExpansion[i]
```

Out[12]= $x^i\ y_i$

Out[13]= $x^i\ y_i$

```
In[14]:= xu[i] yd[i] // ToFlavor[red]
         % // SumExpansion[red@i]
```

Out[14]= $x^i\ y_i$

Out[15]= $x^1\ y_1 + x^2\ y_2 + x^3\ y_3$

A partial expansion can be done over a subset of the base indices. The base subset is never flavored.

```
In[16]:= xu[i] yd[i] // ToFlavor[red]
         % // SumExpansion[red@i, {1, 3}]
```

Out[16]= $x^i\ y_i$

Out[17]= $x^1\ y_1 + x^3\ y_3$

An error occurs if the base specification is not a subset of BaseIndices.

```
In[18]:= xu[i] yd[i] // ToFlavor[red]
         % // SumExpansion[red@i, {1, 4}]
```

Out[18]= $x^i\ y_i$

```
SumArrayExpansion::subset : {1, 4} is not a subset of the base indices {1, 2, 3}
```

```
Out[19]= $Aborted
```

Only the specified indices are expanded.

```
In[20]:= xu[i] yd[i] + xu[j] yd[j]
         % // SumExpansion[j]
```

Out[20]= $x^i\ y_i + x^j\ y_j$

Out[21]= $x^1\ y_1 + x^2\ y_2 + x^3\ y_3 + x^i\ y_i$

It operates on sums on both sides of an equation...

```
In[22]:= xu[i] xd[i] + Sud[i, j] xu[j] <= yu[i] yd[i] + Tud[i, j] yu[j]
         % // SumExpansion[i, j]
```

Out[22]= $S^{i}{}_{j}\, x^{j} + x^{i}\, x_{i} \leq T^{i}{}_{j}\, y^{j} + y^{i}\, y_{i}$

Out[23]= $S^{1}{}_{1}\, x^{1} + S^{2}{}_{1}\, x^{1} + S^{3}{}_{1}\, x^{1} + S^{1}{}_{2}\, x^{2} + S^{2}{}_{2}\, x^{2} + S^{3}{}_{2}\, x^{2} + S^{1}{}_{3}\, x^{3} + S^{2}{}_{3}\, x^{3} + S^{3}{}_{3}\, x^{3} + x^{1}\, x_{1} + x^{2}\, x_{2} + x^{3}\, x_{3} \leq T^{1}{}_{1}\, y^{1} + T^{2}{}_{1}\, y^{1} + T^{3}{}_{1}\, y^{1} + T^{1}{}_{2}\, y^{2} + T^{2}{}_{2}\, y^{2} + T^{3}{}_{2}\, y^{2} + T^{1}{}_{3}\, y^{3} + T^{2}{}_{3}\, y^{3} + T^{3}{}_{3}\, y^{3} + y^{1}\, y_{1} + y^{2}\, y_{2} + y^{3}\, y_{3}$

SumExpansion operates on dot product expressions and CircleTimes expressions.

```
In[24]:= xu[i].yd[i]
         % // SumExpansion[i]
```

Out[24]= $x^{i}.y_{i}$

Out[25]= $x^{1}.y_{1} + x^{2}.y_{2} + x^{3}.y_{3}$

```
In[26]:= xu[i]⊗yd[i]
         % // SumExpansion[i]
```

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

Out[27]= $x^{1} \otimes y_{1} + x^{2} \otimes y_{2} + x^{3} \otimes y_{3}$

It also operates over arrays.

```
In[28]:= {{xu[i] yd[i]}, {xd[i] yu[i]}}
         % // SumExpansion[i]
```

Out[28]= $\{\{x^{i}\, y_{i}\}, \{x_{i}\, y^{i}\}\}$

Out[29]= $\{\{x^{1}\, y_{1} + x^{2}\, y_{2} + x^{3}\, y_{3}\}, \{x_{1}\, y^{1} + x_{2}\, y^{2} + x_{3}\, y^{3}\}\}$

```
In[30]:= xu[x] yd[x]
         % // SumExpansion[x]
```

Out[30]= $x^{x}\, y_{x}$

Out[31]= $x^{1}\, y_{1} + x^{2}\, y_{2} + x^{3}\, y_{3}$

If we have declared special base indices for some flavors of indices, then they are expanded on the corresponding bases.

```
In[32]:= DeclareBaseIndices[{1, 2, 3}, {red, {A, B}}]
```

```
In[33]:= Suu[a, red@b] Tdd[a, red@b]
         % // SumExpansion[a, red@b]
```

Out[33]= $S^{a\, b}\, T_{a\, b}$

Out[34]= $S^{1\, A}\, T_{1\, A} + S^{1\, B}\, T_{1\, B} + S^{2\, A}\, T_{2\, A} + S^{2\, B}\, T_{2\, B} + S^{3\, A}\, T_{3\, A} + S^{3\, B}\, T_{3\, B}$

We can still use selected subsets for each index but, of course, they must be from the corresponding base sets.

```
In[35]:= Suu[a, red@b] Tdd[a, red@b]
         % // SumExpansion[a, red@b, {{1, 2}, {B}}]
```

Out[35]= $S^{a\, b}\, T_{a\, b}$

Out[36]= $S^{1\, B}\, T_{1\, B} + S^{2\, B}\, T_{2\, B}$

Restore the initial values...

```
In[37]:= ClearTensorShortcuts[{{x, y}, 1}, {{S, T}, 2}]

In[38]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

SymbolicIndexQ

■ `SymbolicIndexQ[`*index*`]` returns True if index is a Symbol or if index is `flavor[i_Symbol]` where flavor is a currently active index flavor.

`SymbolicIndexQ` is principally a service routine for programming other routines.

See also: `BaseIndices`, `IndexFlavors`, `RawIndex`.

Examples

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

Save the settings and declare base indices and flavors.

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

The following are symbolic indices...

```
In[7]:= {i, red@i, rocket@j}
        SymbolicIndexQ /@ %

Out[7]= {i, i, j*}

Out[8]= {True, True, True}
```

The following are not symbolic indices...

```
In[9]:= {1, red@2, rocket@2, blue@j, f[k], red@f[k]}
        SymbolicIndexQ /@ %

Out[9]= {1, 2, 2*, blue[j], f[k], f[k]}

Out[10]= {False, False, False, False, False, False}
```

Restore the initial values...

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

SymbolsToPatterns

- `SymbolsToPatterns[`*symbollist*`][`*expr*`]` will convert symbols in expr, which are on symbollist, to named patterns.

This routine is principally used by `LHSSymbolsToPatterns`, the more useful routine. It is a method of changing derived results into general rules that can be used to perform substitutions in further derivations.

See also: `LHSSymbolsToPatterns`.

Examples

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

In[2]:=  oldindices = CompleteBaseIndices;

In[3]:=  DeclareBaseIndices[{0, 1, 2, 3}]
         DefineTensorShortcuts[{u, 1}, {{g, T, G}, 2}]
```

The routine turns symbols into named patterns.

```
In[5]:=  f[x, y]
         % // SymbolsToPatterns[{x, y}]

Out[5]=  f[x, y]

Out[6]=  f[x_, y_]

In[7]:=  f[x, y]
         % // SymbolsToPatterns[{f, x, y}]

Out[7]=  f[x, y]

Out[8]=  f_[x_, y_]
```

The following equation can be changed into a rule.

```
In[9]:=  eqn1 = Tdd[μ, ν] == (ρ + p) ud[μ] ud[ν] - p gdd[μ, ν]
         Print["To make a specific rule"]
         Rule @@ eqn1
         Print["To make a general rule with p and ρ as parameters and μ,ν as patterns"]
         rule1[p_, ρ_] = Rule @@ MapAt[SymbolsToPatterns[{μ, ν}], eqn1, 1]
```

Out[9]= $T_{\mu\nu} == -p\, g_{\mu\nu} + (p + \rho)\, u_\mu\, u_\nu$

```
            To make a specific rule
```

Out[11]= $T_{\mu\nu} \to -p\, g_{\mu\nu} + (p + \rho)\, u_\mu\, u_\nu$

```
            To make a general rule with p and ρ as parameters and μ,ν as patterns
```

Out[13]= $T_{\mu\_\nu\_} \to -p\, g_{\mu\nu} + (p + \rho)\, u_\mu\, u_\nu$

But using `LHSSymbolsToPatterns` is more direct.

This can then be used to substitute in equations.

```
In[14]:= Gdd[α, β] == 8 π Tdd[α, β]
         % /. rule1[p, ρ]
```

Out[14]= $G_{\alpha\beta} == 8\pi\, T_{\alpha\beta}$

Out[15]= $G_{\alpha\beta} == 8\pi\, (-p\, g_{\alpha\beta} + (p + \rho)\, u_\alpha\, u_\beta)$

```
In[16]:= DeclareBaseIndices @@ oldindices
         Clear[eqn1, rule1, oldindices]

In[18]:= ClearTensorShortcuts[{u, 1}, {{g, T, G}, 2}]
```

Symmetric

- `Symmetric[`*indices, weighting : True*`]` `[`*expr*`]` calculates the symmetric tensor expression associated with expr for the list of indices.
- `Symmetric[{`*indices1*`}, {`*indicies2*`}`*..., weighting : True*`]` `[`*expr*`]` uses multiple sets of symmetric indices.

With weighting True (the default value) the `p !` terms generated are divided by `p !` where `p` is the number of antisymmetric indices.

The indices operated on in each set must be all up or all down.

The indices in the command must carry the flavor.

Symmetric is automatically mapped over arrays, equations and sums.

See also: `AntiSymmetric`, `IndexChange`, `SymmetrizeSlots`, `SymmetrizePattern`.

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[{{T, S, A}, 2}, {T, 3}]
```

```
In[6]:= Tuu[i, j]
        % // Symmetric[{i, j}]
```

Out[6]= T^{ij}

Out[7]= $\frac{1}{2}\left(T^{ij}+T^{ji}\right)$

The indices in the command must carry their flavors.

```
In[8]:= Tuu[i, j] // ToFlavor[red]
        % // Symmetric[red /@ {i, j}]
```

Out[8]= T^{ij}

Out[9]= $\frac{1}{2}\left(T^{ij}+T^{ji}\right)$

Mixed indicies can't be symmetrized.

```
In[10]:= Tudd[i, j, k]
         % // Symmetric[{i, j, k}]
```

Out[10]= $T^{i}{}_{jk}$

```
Symmetric::indices : Symmetric indices {i, j, k} must be all be in
   the list of up, {i}, or all in the list of down, {j, k}, indices.
```

```
Out[11]= $Aborted
```

Here we give T numerical values.

In[12]:= `SetTensorValues[Tdd[i, j],` $\begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & -5 \\ 3 & -3 & 2 \end{pmatrix}$`]`

This defines S to the the symmetric part.

```
In[13]:= Tdd[i, j]
         % // Symmetric[{i, j}]
         % // EinsteinArray[] // MatrixForm
         SetTensorValues[Sdd[i, j], %]
```

Out[13]= T_{ij}

Out[14]= $\frac{1}{2}\left(T_{ij} + T_{ji}\right)$

Out[15]//MatrixForm=

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & -4 \\ 3 & -4 & 2 \end{pmatrix}$$

This defines A to be the antisymmetric part.

```
In[17]:= Tdd[i, j]
         % // AntiSymmetric[{i, j}]
         % // EinsteinArray[] // MatrixForm
         SetTensorValues[Add[i, j], %]
```

Out[17]= T_{ij}

Out[18]= $\frac{1}{2}\left(T_{ij} - T_{ji}\right)$

Out[19]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

They sum to the original tensor.

```
In[21]:= Sdd[i, j] + Add[i, j]
         % // EinsteinArray[] // MatrixForm
```

Out[21]= $A_{ij} + S_{ij}$

Out[22]//MatrixForm=

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & -5 \\ 3 & -3 & 2 \end{pmatrix}$$

Higher order tensors can also be symmetrized.

```
In[23]:= Tddd[i, j, k]
         % // Symmetric[{i, j, k}]
```

Out[23]= T_{ijk}

Out[24]= $\frac{1}{6}\left(T_{ijk}+T_{ikj}+T_{jik}+T_{jki}+T_{kij}+T_{kji}\right)$

```
In[25]:= Tdd[i, j] Sdd[m, n]
         % // Symmetric[{i, m}, {j, n}]
```

Out[25]= $S_{mn}\,T_{ij}$

Out[26]= $\frac{1}{2}\left(\frac{1}{2}S_{mn}\,T_{ij}+\frac{1}{2}S_{mj}\,T_{in}\right)+\frac{1}{2}\left(\frac{1}{2}S_{in}\,T_{mj}+\frac{1}{2}S_{ij}\,T_{mn}\right)$

With the optional argument weighting set to False, no weighting factors are included.

```
In[27]:= Tdd[i, j] Sdd[m, n]
         % // Symmetric[{i, m}, {j, n}, False]
```

Out[27]= $S_{mn}\,T_{ij}$

Out[28]= $S_{mn}\,T_{ij}+S_{mj}\,T_{in}+S_{in}\,T_{mj}+S_{ij}\,T_{mn}$

Selected indices can, of course, be symmetrized.

```
In[29]:= Tdd[i, j] Sdd[m, n] // ToFlavor[red]
         % // Symmetric[red /@ {i, n}]
```

Out[29]= $S_{mn}\,T_{ij}$

Out[30]= $\frac{1}{2}\left(S_{mn}\,T_{ij}+S_{mi}\,T_{nj}\right)$

Symmetric and SymmetrizePattern with the same symmetry are inverse operations.

```
In[31]:= Tdd[i, j] Sdd[m, n] // ToFlavor[red]
         % // Symmetric[red /@ {i, n}]
         % // SymmetrizePattern[Sdd[_, b_] Tdd[a_, _], {{1, {a, b}}}]
```

Out[31]= $S_{mn}\,T_{ij}$

Out[32]= $\frac{1}{2}\left(S_{mn}\,T_{ij}+S_{mi}\,T_{nj}\right)$

Out[33]= $S_{mn}\,T_{ij}$

Restore the settings

```
In[34]:= ClearTensorValues /@ {Tdd[i, j], Sdd[i, j], Add[i, j]};
         ClearTensorShortcuts[{{T, S, A}, 2}, {T, 3}]
```

```
In[36]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

SymmetrizePattern

- `SymmetrizePattern[`*patternl, symmetries*`][`*expr*`]` will permute the (anti)symmetric indices in all terms matching the pattern such that they are ordered according to the pattern and the specified symmetries.
- `SymmetrizePattern[][`*expr*`]` will use the symmetries defined by `DeclarePatternSymmetries`.

The symmetries are given by a list of symmetry specifications, `{symm1, symm2, ...}`.

Each symmetry specification is of the form `{-1 | 0 | 1, {a, b, ...}}` where `-1` is used for antisymmetry and 1 for symmetry. a, b, etc., are the are the index pattern names in the pattern.

It is also possible to specify equal length groups of indices that can be interchanged without internal reordering. This would be done in the form `{-1 | 0 | 1, {{a, b}, {c, d}, ...}}`.

The order of the named indices in the symmetry specifications does not matter - they are sorted. But the placement of the named indices in the pattern does matter and determines the canonical form.

By putting indices in standard order with multiple terms, *Mathematica* will perform simplifications.

SymmetrizeSlots works on individual tensors by slot position. SymmetrizePatterns will work on more complex terms involving multiple tensors, derivatives, CircleTimes or Dot. The essential requirement is that the pattern must describe a term that ParseTerm-Indices will parse.

The symmetry routines in Tensorial are much more general than the round and square bracket notation sometimes used in texts because it allows grouping of indices and it also allows nonadjacent indices to be associated.

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

Examples

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

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DefineTensorShortcuts[{{e, x}, 1}, {S, 2}, {{S, T}, 3}, {R, 4}, {R, 6}]
        DeclareIndexFlavor[{red, Red}]
        labs = {x, δ, g, Γ};
```

Here we have symmetries that span several tensors. The upper and lower indices are each symmetric. The two terms are put in the same index form and combined. In these examples the pattern is usually shown as the first line.

```
In[8]:= Suud[a_, b_, d_] Tudd[c_, e_, f_]
        Suud[a, b, d] Tudd[c, e, f] + Suud[b, a, d] Tudd[c, f, e]
        % // SymmetrizePattern[%%, {{1, {a, b, c}}, {1, {d, e, f}}}]
```

Out[8]= $S^{a\_\,b\_}{}_{d\_}\,T^{c\_}{}_{e\_\,f\_}$

Out[9]= $S^{a\,b}{}_{d}\,T^{c}{}_{e\,f} + S^{b\,a}{}_{d}\,T^{c}{}_{f\,e}$

Out[10]= $2\,S^{a\,b}{}_{d}\,T^{c}{}_{e\,f}$

The color in the expression does not have to be in the pattern.

```
In[11]:= Suud[a_, b_, d_] Tudd[c_, e_, f_]
         Suud[a, b, d] Tudd[c, e, f] + Suud[b, a, d] Tudd[c, f, e] // ToFlavor[red]
         % // SymmetrizePattern[%%, {{1, {a, b, c}}, {1, {d, e, f}}}]
```

Out[11]= $S^{a\_b\_}{}_{d\_}\, T^{c\_}{}_{e\_f\_}$

Out[12]= $S^{ab}{}_{d}\, T^{c}{}_{ef} + S^{ba}{}_{d}\, T^{c}{}_{fe}$

Out[13]= $2\, S^{ab}{}_{d}\, T^{c}{}_{ef}$

```
In[14]:= Suud[red@a_, red@b_, red@d_] Tudd[red@c_, red@e_, red@f_]
         Suud[a, b, d] Tudd[c, e, f] + Suud[b, a, d] Tudd[c, f, e] // ToFlavor[red]
         % // SymmetrizePattern[%%, {{1, {a, b, c}}, {1, {d, e, f}}}]
```

Out[14]= $S^{a\_b\_}{}_{d\_}\, T^{c\_}{}_{e\_f\_}$

Out[15]= $S^{ab}{}_{d}\, T^{c}{}_{ef} + S^{ba}{}_{d}\, T^{c}{}_{fe}$

Out[16]= $2\, S^{ab}{}_{d}\, T^{c}{}_{ef}$

In the following case the bottom indices are defined to be antisymmetric. The antisymmetry is indicated by the `-1` in the second group of indices. The two terms cancel if the number of transpositions to canonical sort order is odd.

```
In[17]:= Suud[a_, b_, d_] Tudd[c_, e_, f_]
         Suud[a, b, d] Tudd[c, e, f] + Suud[b, a, d] Tudd[c, f, e]
         % // SymmetrizePattern[%%, {{1, {a, b, c}}, {-1, {d, e, f}}}]
```

Out[17]= $S^{a\_b\_}{}_{d\_}\, T^{c\_}{}_{e\_f\_}$

Out[18]= $S^{ab}{}_{d}\, T^{c}{}_{ef} + S^{ba}{}_{d}\, T^{c}{}_{fe}$

Out[19]= 0

The pattern index names do not have to be the same as the index names in the actual expression.

```
In[20]:= Suud[a_, b_, d_] Tudd[c_, e_, f_]
         Suud[α, β, δ] Tudd[γ, ϵ, ϕ] + Suud[β, α, δ] Tudd[γ, ϕ, ϵ]
         % // SymmetrizePattern[%%, {{1, {a, b, c}}, {1, {d, e, f}}}]
```

Out[20]= $S^{a\_b\_}{}_{d\_}\, T^{c\_}{}_{e\_f\_}$

Out[21]= $S^{\alpha\beta}{}_{\delta}\, T^{\gamma}{}_{\epsilon\phi} + S^{\beta\alpha}{}_{\delta}\, T^{\gamma}{}_{\phi\epsilon}$

Out[22]= $2\, S^{\alpha\beta}{}_{\delta}\, T^{\gamma}{}_{\epsilon\phi}$

The canonical form is determined by the placement of the pattern names in the pattern. Here the order of the upper indices is reversed. In this case only two of the upper indices are included in the symmetry. Since `{c, b}` are in reverse sort order, so are `{β, α}`.

```
In[23]:= Suud[c_, b_, d_] Tudd[_, e_, f_]
         Suud[α, β, δ] Tudd[γ, ϵ, ϕ] + Suud[β, α, δ] Tudd[γ, ϕ, ϵ]
         % // SymmetrizePattern[%%, {{1, {b, c}}, {1, {d, e, f}}}]
```

Out[23]= $S^{c\_\,b\_}{}_{d\_}\,T^{\_}{}_{e\_\,f\_}$

Out[24]= $S^{\alpha\beta}{}_{\delta}\,T^{\gamma}{}_{\epsilon\phi} + S^{\beta\alpha}{}_{\delta}\,T^{\gamma}{}_{\phi\epsilon}$

Out[25]= $2\,S^{\beta\alpha}{}_{\delta}\,T^{\gamma}{}_{\epsilon\phi}$

Here is a covariant derivative case

```
In[26]:= CovariantD[Suu[a_, b_], d_] Tudd[c_, e_, f_]
         CovariantD[Suu[α, β], δ] Tudd[γ, ϵ, ϕ] + CovariantD[Suu[β, α], δ] Tudd[γ, ϕ, ϵ]
         % // SymmetrizePattern[%%, {{1, {a, b, c}}, {-1, {d, e, f}}}]
```

Out[26]= $S^{a\_\,b\_}{}_{;d\_}\,T^{c\_}{}_{e\_\,f\_}$

Out[27]= $S^{\alpha\beta}{}_{;\delta}\,T^{\gamma}{}_{\epsilon\phi} + S^{\beta\alpha}{}_{;\delta}\,T^{\gamma}{}_{\phi\epsilon}$

Out[28]= 0

Here is a case with an expanded partial derivative. We need two patterns to reflect the symmetries in all of the terms.

```
In[29]:= PartialD[labs][Suud[a_, b_, d_], xu[_]] Tudd[c_, e_, f_] |
          PartialD[labs][Tudd[c_, e_, f_], xu[_]] Suud[a_, b_, d_]
         Suud[α, β, δ] Tudd[γ, ϵ, ϕ] + Suud[β, α, δ] Tudd[γ, ϕ, ϵ] // ToFlavor[red]
         PartialD[labs][%, xu[red@a]]
         % // SymmetrizePattern[%%%, {{1, {a, b, c}}, {1, {d, e, f}}}]
```

Out[29]= $T^{c\_}{}_{e\_\,f\_}\,\frac{\partial S^{a\_\,b\_}{}_{d\_}}{\partial x\_} \;\Big|\; S^{a\_\,b\_}{}_{d\_}\,\frac{\partial T^{c\_}{}_{e\_\,f\_}}{\partial x\_}$

Out[30]= $S^{\alpha\beta}{}_{\delta}\,T^{\gamma}{}_{\epsilon\phi} + S^{\beta\alpha}{}_{\delta}\,T^{\gamma}{}_{\phi\epsilon}$

Out[31]= $T^{\gamma}{}_{\epsilon\phi}\,\frac{\partial S^{\alpha\beta}{}_{\delta}}{\partial x^{a}} + T^{\gamma}{}_{\phi\epsilon}\,\frac{\partial S^{\beta\alpha}{}_{\delta}}{\partial x^{a}} + S^{\alpha\beta}{}_{\delta}\,\frac{\partial T^{\gamma}{}_{\epsilon\phi}}{\partial x^{a}} + S^{\beta\alpha}{}_{\delta}\,\frac{\partial T^{\gamma}{}_{\phi\epsilon}}{\partial x^{a}}$

Out[32]= $2\,T^{\gamma}{}_{\epsilon\phi}\,\frac{\partial S^{\alpha\beta}{}_{\delta}}{\partial x^{a}} + 2\,S^{\alpha\beta}{}_{\delta}\,\frac{\partial T^{\gamma}{}_{\epsilon\phi}}{\partial x^{a}}$

CircleTimes and Dot products can also be processed.

```
In[33]:= 1 / 2 (ed[i].ed[j] + ed[j].ed[i])
         % // SymmetrizePattern[ed[a_].ed[b_], {{1, {a, b}}}]
```

Out[33]= $\frac{1}{2}\left(e_i . e_j + e_j . e_i\right)$

Out[34]= $e_i . e_j$

The Riemann tensor has a symmetry that allows the interchange of the first two indices with the last two. We specify this by enclosing each group in brackets.

```
In[35]:= Rdddd[γ, δ, α, β] + Rdddd[α, β, γ, δ]
         % // SymmetrizePattern[Rdddd[a_, b_, c_, d_], {{1, {{a, b}, {c, d}}}}]
```

Out[35]= $R_{\alpha\beta\gamma\delta} + R_{\gamma\delta\alpha\beta}$

Out[36]= $2\ R_{\alpha\beta\gamma\delta}$

There is no internal sorting of the groups. That could be specified with additional symmetries.

```
In[37]:= Rdddd[γ, δ, β, α] + Rdddd[β, α, γ, δ]
         % // SymmetrizePattern[Rdddd[a_, b_, c_, d_], {{1, {{a, b}, {c, d}}}}]
         % // SymmetrizePattern[Rdddd[a_, b_, _, _], {{-1, {a, b}}}]
```

Out[37]= $R_{\beta\alpha\gamma\delta} + R_{\gamma\delta\beta\alpha}$

Out[38]= $2\ R_{\beta\alpha\gamma\delta}$

Out[39]= $-2\ R_{\alpha\beta\gamma\delta}$

The following hypothetical tensor has 3 antisymmetric groups of 2 indices each. The second term has an odd number of transpositions and enters with a negative sign.

```
In[40]:= Rdddddd[a, b, c, d, e, f] + Rdddddd[c, d, a, b, e, f] + Rdddddd[c, d, e, f, a, b]
         % //
          SymmetrizePattern[Rdddddd[a_, b_, c_, d_, e_, f_], {{-1, {{a, b}, {c, d}, {e, f}}}}]
```

Out[40]= $R_{abcdef} + R_{cdabef} + R_{cdefab}$

Out[41]= R_{abcdef}

It would enter with a positive sign if we had no grouping.

```
In[42]:= Rdddddd[a, b, c, d, e, f] + Rdddddd[c, d, a, b, e, f] + Rdddddd[c, d, e, f, a, b]
         % // SymmetrizePattern[Rdddddd[a_, b_, c_, d_, e_, f_], {{-1, {a, b, c, d, e, f}}}]
```

Out[42]= $R_{abcdef} + R_{cdabef} + R_{cdefab}$

Out[43]= $3\ R_{abcdef}$

In the following case we have a regular symmetry for the upper indices and a grouping symmetry for the lower indices.

```
In[44]:= Ruudddd[a, b, c, d, e, f] + Ruudddd[b, a, c, d, e, f] + Ruudddd[a, b, e, f, c, d] //
          ToFlavor[red]
         % // SymmetrizePattern[Ruudddd[a_, b_, c_, d_, e_, f_],
           {{1, {{c, d}, {e, f}}}, {1, {a, b}}}]
```

Out[44]= $R^{ab}{}_{cdef} + R^{ab}{}_{efcd} + R^{ba}{}_{cdef}$

Out[45]= $3\ R^{ab}{}_{cdef}$

In the following case the second term is rearranged by groups but does not combine with the other terms because there is no rearrangement within the groups.

```
In[46]:= Ruudddd[a, b, c, d, e, f] + Ruudddd[b, a, c, d, e, f] + Ruudddd[a, b, e, f, d, c] //
          ToFlavor[red]
         % // SymmetrizePattern[Ruudddd[a_, b_, c_, d_, e_, f_],
           {{1, {{c, d}, {e, f}}}, {1, {a, b}}}]
```

Out[46]= $R^{ab}{}_{cdef} + R^{ab}{}_{efdc} + R^{ba}{}_{cdef}$

Out[47]= $2\,R^{ab}{}_{cdef} + R^{ab}{}_{dcef}$

See the examples in `DeclarePatternSymmetries` for examples using `SymmetrizePattern[]`.

Restore the original state.

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

```
In[52]:= ClearTensorShortcuts[{{e, x}, 1}, {S, 2}, {{S, T}, 3}, {R, 4}, {R, 6}]
```

SymmetrizeSlots

- `SymmetrizeSlots[`*label, order, symmetry*`][`*expr*`]` will put indices in standard sort order for all tensors of the given label and order in expr.
- `SymmetrizeSlots[][`*expr*`]` will use the symmetries defined by `TensorSymmetry` definitions.

The symmetry is defined as in the `TensorSymmetry` Help page.

By putting indices in standard sort order with multiple terms, *Mathematica* will perform simplifications.

Identical indices in antisymmetrical slots will cause the tensor to be set to zero.

`SymmetrizePattern` or `IndexChange` can be used to impose symmetries in a multi-tensor term.

See also: `TensorSymmetry`, `SymmetrizePattern`, `Symmetric`, `AntiSymmetric`, `IndexChange`.

Examples

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

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

Here we simplify a sum when `T` is symmetric or antisymmetric in its indices.

```
In[3]:= Tuu[i, j] + Tuu[j, i]
        Print["Symmetric case"]
        %% // SymmetrizeSlots[T, 2, Symmetric[1, 2]]
        Print["Antisymmetric case"]
        %%%% // SymmetrizeSlots[T, 2, AntiSymmetric[1, 2]]
```

Out[3]= $T^{ij} + T^{ji}$

```
   Symmetric case
```

Out[5]= $2\,T^{ij}$

```
   Antisymmetric case
```

Out[7]= 0

The following examples show how various symmetries affect a contracted tensor.

```
In[8]:= ωuud[b, a, b]
        Print["Symmetric in the first and second slots"]
        %% // SymmetrizeSlots[ω, 3, Symmetric[1, 2]]
```

Out[8]= $\omega^{ba}{}_{b}$

```
   Symmetric in the first and second slots
```

Out[10]= $\omega^{ab}{}_{b}$

```
In[11]:= ωuud[b, a, b]
         Print["Anti-symmetric in the first and second slots"]
         %% // SymmetrizeSlots[ω, 3, AntiSymmetric[1, 2]]
```

Out[11]= $\omega^{b\,a}{}_{b}$

```
           Anti-symmetric in the first and second slots
```

Out[13]= $-\omega^{a\,b}{}_{b}$

```
In[14]:= ωuud[b, a, b]
         Print["Anti-symmetric in the second and third slots"]
         %% // SymmetrizeSlots[ω, 3, AntiSymmetric[2, 3]]
```

Out[14]= $\omega^{b\,a}{}_{b}$

```
           Anti-symmetric in the second and third slots
```

Out[16]= $\omega^{b\,a}{}_{b}$

```
In[17]:= ωuud[b, c, a]
         Print["Symmetric in the first and third slots"]
         %% // SymmetrizeSlots[ω, 3, Symmetric[1, 3]]
```

Out[17]= $\omega^{b\,c}{}_{a}$

```
           Symmetric in the first and third slots
```

Out[19]= $\omega_{a}{}^{c\,b}$

```
In[20]:= ωuud[b, a, b]
         Print["Anti-symmetric in the first and third slots"]
         %% // SymmetrizeSlots[ω, 3, AntiSymmetric[1, 3]]
```

Out[20]= $\omega^{b\,a}{}_{b}$

```
           Anti-symmetric in the first and third slots
```

Out[22]= 0

```
In[23]:= ωuud[b, a, b]
         Print["Anti-symmetric in all the slots"]
         %% // SymmetrizeSlots[ω, 3, AntiSymmetric[1, 2, 3]]
```

Out[23]= $\omega^{b\,a}{}_{b}$

```
           Anti-symmetric in all the slots
```

Out[25]= 0

The following illustrates the operation on base indices. First with symmetric indices.

```
In[26]:= Tdd[i, j]
         % // ToArrayValues[] // MatrixForm
         % // SymmetrizeSlots[T, 2, Symmetric[1, 2]] // MatrixForm
```

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

Out[27]//MatrixForm=

$$\begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix}$$

Out[28]//MatrixForm=

$$\begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{12} & T_{22} & T_{23} \\ T_{13} & T_{23} & T_{33} \end{pmatrix}$$

Then with antisymmetric indices.

```
In[29]:= Tdd[i, j]
         % // ToArrayValues[] // MatrixForm
         % // SymmetrizeSlots[T, 2, AntiSymmetric[1, 2]] // MatrixForm
```

Out[29]= $T_{i j}$

Out[30]//MatrixForm=

$$\begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix}$$

Out[31]//MatrixForm=

$$\begin{pmatrix} 0 & T_{12} & T_{13} \\ -T_{12} & 0 & T_{23} \\ -T_{13} & -T_{23} & 0 \end{pmatrix}$$

```
In[32]:= ClearTensorShortcuts[{T, 2}, {ω, 3}]
```

In[28]:=

```
See also the examples in TensorSymmetry for examples of defined tensor symmetries.
```