IndexChange

- `IndexChange[{{r, i}, {s, j}...}, sign : 1][expr]` will replace the first index in each pair list by the second index in the pair list everywhere in the expression and multiply the expression by sign, which has the default value of 1.
- `IndexChange[{r, i}, sign : 1][expr]` may be used for a single index change.

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

IndexChange works only on symbolic indices and not base indices

Each tensor term is checked to see if it contains all the indices that are to be replaced (the first indices in each pair). It it does not then no change is made to that term. Expressions may have to be Expanded by the user to bring all indices together in a term.

IndexChange requires the flavor on the indices.

TensorSimplify or SimplifyTensorSum will perform automatic reindexing in an attempt to simplify tensor expressions.

See also: `EinsteinSum`, `SimplifyTensorSum`, `DummySimplify`, `SymmetrizeSlots`.

Examples

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

Save old values and declare a flavor.

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

```
In[5]:= DefineTensorShortcuts[{{A, B, x}, 1}, {{A, B}, 2}, {A, 3}]
```

SimplifyTensorSum will usually perform the desired dummy index simplification in expressions.

```
In[6]:= expr1 = a Ad[j] Bu[j] + b Ad[m] Bu[m] + c Ad[i] Bu[i]
        % // SimplifyTensorSum // Factor
```

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

Out[7]= $(a + b + c)\,A_i\,B^i$

But we could also do it "by hand". We must write separate commands for each term because no term contains two indices.

```
In[8]:= expr1
        % // IndexChange[{j, i}] // IndexChange[{m, i}] // Factor
```

Out[8]= $c\,A_i\,B^i + a\,A_j\,B^j + b\,A_m\,B^m$

Out[9]= $(a + b + c)\,A_i\,B^i$

We could also write a sequence of index changes in the following manner.

```
In[10]:= expr1
         Fold[IndexChange[#2][#1] &, %, {{j, i}, {m, i}}] // Factor
```

Out[10]= $c\, A_i\, B^i + a\, A_j\, B^j + b\, A_m\, B^m$

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

The index must carry the flavor.

```
In[12]:= expr1 // ToFlavor[red]
         % // IndexChange[red /@ {j, i}] // IndexChange[red /@ {m, i}] // Factor
```

Out[12]= $c\, A_i\, B^i + a\, A_j\, B^j + b\, A_m\, B^m$

Out[13]= $(a + b + c)\, A_i\, B^i$

The following modifies the second term. The first term does not contain both i and j and so is left unmodified.

```
In[14]:= Au[i] Ad[i] + Buu[i, j] Bdd[i, j]
         % // IndexChange[{{i, j}, {j, i}}]
```

Out[14]= $A^i\, A_i + B^{i\,j}\, B_{i\,j}$

Out[15]= $A^j\, A_j + B^{j\,i}\, B_{j\,i}$

Here both terms contain i and j and so both terms are modified.

```
In[16]:= Auu[j, i] Add[j, i] + Buu[i, j] Bdd[i, j]
         % // IndexChange[{{i, j}, {j, i}}]
```

Out[16]= $A^{j\,i}\, A_{j\,i} + B^{i\,j}\, B_{i\,j}$

Out[17]= $A^{i\,j}\, A_{i\,j} + B^{j\,i}\, B_{j\,i}$

Here we implement an antisymmetry in a tensor expression by specifying the sign as -1.

```
In[18]:= Add[i, j] Bdd[k, l]
         % // IndexChange[{{j, k}, {k, j}}, -1]
```

Out[18]= $A_{i\,j}\, B_{k\,l}$

Out[19]= $-A_{i\,k}\, B_{j\,l}$

The use of a symbol in a tensor label or as a constant is distinguished from its use in a tensor index. Only the index is changed.

```
In[20]:= A Add[i, A] Bdd[B, l]
         % // IndexChange[{{A, B}, {B, A}}, -1]
```

Out[20]= $A\, A_{i\,A}\, B_{B\,l}$

Out[21]= $-A\, A_{i\,B}\, B_{A\,l}$

The following commands substitute the second set of indices for the first set of indices.

```
In[22]:= Auuu[i, j, k]
         % // IndexChange[{{i, j, k}, {μ, ν, λ}} // Transpose]
```

Out[22]= $A^{i\,j\,k}$

Out[23]= $A^{\mu\,\nu\,\lambda}$

Or we could rotate a set of indices with the following command.

```
In[24]:= Auuu[i, j, k]
         % // IndexChange[{idxs = {i, j, k}, RotateLeft[idxs]} // Transpose]
```

Out[24]= $A^{i\,j\,k}$

Out[25]= $A^{j\,k\,i}$

```
In[26]:= Au[a] PartialD[{x, δ, g, Γ}][Bud[b, c], xu[a]]
         % // IndexChange[{{a, b}, {b, a}}]
```

Out[26]= $A^{a}\,\frac{\partial B^{b}{}_{c}}{\partial x^{a}}$

Out[27]= $A^{b}\,\frac{\partial B^{a}{}_{c}}{\partial x^{b}}$

```
In[28]:= {Au[a] PartialD[Bud[b, c], a], Au[a] CovariantD[Bud[b, c], {a, d}]}
         % // IndexChange[{{a, b}, {b, a}}]
```

Out[28]= $\{B^{b}{}_{c,a}\,A^{a},\ B^{b}{}_{c;a\,d}\,A^{a}\}$

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

On partial and covariant derivatives.

```
In[30]:= {PartialD[Au[a], b], CovariantD[Au[a], b]}
         % // IndexChange[{b, k}]
         % // IndexChange[{{a, r}, {k, s}}]
```

Out[30]= $\{A^{a}{}_{,b},\ A^{a}{}_{;b}\}$

Out[31]= $\{A^{a}{}_{,k},\ A^{a}{}_{;k}\}$

Out[32]= $\{A^{r}{}_{,s},\ A^{r}{}_{;s}\}$

```
In[33]:= PartialD[Au[a], b]
         % // ExpandPartialD[{x, δ, g, Γ}]
         % // IndexChange[{{a, r}, {b, s}}]
```

Out[33]= $A^{a}{}_{,b}$

Out[34]= $\frac{\partial A^{a}}{\partial x^{b}}$

Out[35]= $\frac{\partial A^{r}}{\partial x^{s}}$

```
In[36]:= CovariantD[Au[a], b]
         % // ExpandCovariantD[{x, δ, g, Γ}, c]
         % // IndexChange[{{a, r}, {b, s}}]
```

Out[36]= $A^{a}{}_{;b}$

Out[37]= $A^{c}\,\Gamma^{a}{}_{b\,c} + \frac{\partial A^{a}}{\partial x^{b}}$

Out[38]= $A^{c}\,\Gamma^{r}{}_{s\,c} + \frac{\partial A^{r}}{\partial x^{s}}$

Restore settings

```
In[39]:= ClearTensorShortcuts[{{A, B}, 1}, {{A, B}, 2}, {A, 3}]
```

```
In[40]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[expr1, oldflavors, idxs]
```

IndexFlavorQ

■ `IndexFlavorQ[`*symbol*`]` returns True if symbol is a flavorname in the IndexFlavors list and False otherwise

This is principally a service routine for programming other routines.

See also: `IndexFlavors`, `DeclareIndexFlavor`, `ClearIndexFlavor`, `ToFlavor`.

Examples

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

The following saves the current state.

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

This declares a set of index flavors...

```
In[4]:= DeclareIndexFlavor[{red, Red}, {green, Cerulean}, {rocket, SuperStar}];
```

```
In[5]:= IndexFlavorQ /@ {red, green, rocket, blue, lab}

Out[5]= {True, True, True, False, False}
```

This resets to the original state.

```
In[6]:= ClearIndexFlavor[IndexFlavors];
        DeclareIndexFlavor /@ oldflavors;
        Clear[oldflavors]
```

IndexFlavors

- `IndexFlavors` gives the current set of index flavors and their forms.

Index flavors are used to distinguish various coordinate systems or reference frames.

"The bars, primes, and hats [on the indices] distinguish one coordinate system from another: by putting them on the indices rather than on [the labels] we simplify later notation." Charles W. Misner, Kip S. Thorne & John Archibald Wheeler, *Gravitation*, p9.

According to J. Forster & J.D. Nightingale in *A Short Course in General Relativity*, this "is part of the *kernel-index method* initiated by Shouten and his co-workers." (Schouten, J.A. (1954) *Ricci-Calculus*, 2nd ed., Springer, Berlin)

See the entries under `DeclareIndexFlavor` for further details of specifying flavors.

See also: `DeclareIndexFlavor`, `DeclareBaseIndices`, `ClearIndexFlavor`, `ToFlavor`, `IndexFlavorQ`.

Examples

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

The following gives the current set of of index flavors.

```
In[2]:= oldflavors = IndexFlavors

Out[2]= {}
```

This declares new index flavors.

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

Out[4]= {{red, RGBColor[1, 0, 0]}, {rocket, SuperStar}}
```

This resets to the old indices.

```
In[5]:= ClearIndexFlavor /@ IndexFlavors;
        DeclareIndexFlavor /@ oldflavors;
        Clear[oldflavors]
```

IndexParsingRules

- `IndexParsingRules` is a supplemental list of rules that will be used in `ParseTermIndices` to extract the dummy, up and down indices from a tensor term.

`IndexParsingRules` is initially set to `{}` and should be reset to this value if any supplemental rules are to be removed. Any new setting completely replaces the old setting.

`IndexParsingRules` is used internally by `ParseTermIndices` to extract tensors whose indices are to be parsed from an expression. Its purpose is to allow expressions or constructions whose form is not internally recognized by Tensorial. Generally you will want to convert the expression to a Tensor or a product of Tensors, which will then be correctly parsed.

If a symbol from a subsidiary package is used, then the full context name of the symbol should be used in any rules.

See also: `ParseTermIndices`, `EinsteinSum`, `EinsteinArray`, `ExtractFreeIndices`.

Examples

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

Save the old settings.

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

```
In[8]:= DefineTensorShortcuts[{{e, f}, 1}, {{T}, 2}]
```

The following expression won't expand to a set of equations because `ParseTermIndices` will not look inside the `f` expression. It could not find the free index on the left hand side.

```
In[9]:= f[ed[i]] == Tdu[i, j] fd[j]
        % // EinsteinArray[]
```

Out[9]= $f[e_i] == f_j\, T_i{}^j$

```
FreeIndices::notmatched : The free indices are not the same
   in all terms of the expression or some terms have bad indices.
```

```
Out[10]= $Aborted
```

We can change Tensorial's behavior by adding a supplementary rule for parsing term indices.

```
In[11]:= IndexParsingRules = {f[t_Tensor] → t};
```

Now, Tensorial will parse the f expression and expand the equations.

```
In[12]:= f[ed[i]] == Tdu[i, j] fd[j]
         % // EinsteinArray[] // TableForm
```

Out[12]= $f[e_i] == f_j\, T_i{}^j$

Out[13]//TableForm=

$f[e_1] == f_j\, T_1{}^j$

$f[e_2] == f_j\, T_2{}^j$

$f[e_3] == f_j\, T_3{}^j$

`AngleBracket` is not normally recognized by `ParseTermIndices` but we can add a rule to recognize it.

```
In[14]:= IndexParsingRules = {};
         term1 = 〈eu[a], ed[b]〉
         % // ParseTermIndices
         IndexParsingRules = 〈t1_, t2_〉 → t1 t2
         term1 // ParseTermIndices
         term1 // EinsteinArray[] // MatrixForm
```

Out[15]= $\langle e^a, e_b\rangle$

Out[16]= {{}, {{}, {}}, {}}

Out[17]= 〈t1_, t2_〉 → t1 t2

Out[18]= {{}, {{a}, {b}}, {}}

Out[19]//MatrixForm=

$$\begin{pmatrix} \langle e^1, e_1\rangle & \langle e^1, e_2\rangle & \langle e^1, e_3\rangle \\ \langle e^2, e_1\rangle & \langle e^2, e_2\rangle & \langle e^2, e_3\rangle \\ \langle e^3, e_1\rangle & \langle e^3, e_2\rangle & \langle e^3, e_3\rangle \end{pmatrix}$$

Restore the initial settings...

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

```
In[21]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         IndexParsingRules = oldparsingrules;
         Clear[oldindices, oldflavors, oldparsingrules, term1]
```

KroneckerAbsorb

■ `KroneckerAbsorb[δ][expr]` will perform all replacement operations of first order mixed tensors with labels δ, assumed to be Kronecker deltas.

In Tensorial all Kronecker symbols must have one Void in each slot just like all other indexed objects. Many texts use indices in both the up and down positions, taking advantage of the fact that Kroneckers must be even order with equal number of up and down indices.

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

KroneckerAbsorb will effective reach into any expressions that ParseTermIndices will reach into.

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

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}};
```

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

```
In[8]:= δud[i, r] δud[r, j]
        % // KroneckerAbsorb[δ]
```

Out[8]= $\delta^{i}{}_{r}\,\delta^{r}{}_{j}$

Out[9]= $\delta^{i}{}_{j}$

```
In[10]:= δud[i, r] δud[r, j] // ToFlavor[red]
         % // KroneckerAbsorb[δ]
```

Out[10]= $\delta^{i}{}_{r}\,\delta^{r}{}_{j}$

Out[11]= $\delta^{i}{}_{j}$

```
In[12]:= δud[i, r] δud[r, s] δud[s, j]
         % // KroneckerAbsorb[δ]
```

Out[12]= $\delta^{i}{}_{r}\,\delta^{r}{}_{s}\,\delta^{s}{}_{j}$

Out[13]= $\delta^{i}{}_{j}$

```
In[14]:= xu[j] δud[r, j]
         % // KroneckerAbsorb[δ]
```

Out[14]= $x^j \delta^r{}_j$

Out[15]= x^r

```
In[16]:= δud[j, r] Sud[k, j]
         % // KroneckerAbsorb[δ]
```

Out[16]= $S^k{}_j \delta^j{}_r$

Out[17]= $S^k{}_r$

KroneckerAbsorb will work on mixed flavor tensors, but the flavors must match.

```
In[18]:= {δud[red@j, red@r] Sud[k, red@j], δud[j, r] Sud[k, red@j], δud[j, r] Sud[r, red@j]}
         % // KroneckerAbsorb[δ]
```

Out[18]= $\{S^k{}_j \delta^j{}_r, S^k{}_j \delta^j{}_r, S^r{}_j \delta^j{}_r\}$

Out[19]= $\{S^k{}_r, S^k{}_j \delta^j{}_r, S^j{}_j\}$

```
In[20]:= δud[j, r] CovariantD[Su[k], j] // ToFlavor[red]
         % // KroneckerAbsorb[δ]
```

Out[20]= $S^k{}_{;j} \delta^j{}_r$

Out[21]= $S^k{}_{;r}$

The Kronecker can be represented by a label other than δ.

```
In[22]:= κud[j, r] Sud[k, j] // ToFlavor[rocket]
         % // KroneckerAbsorb[κ]
```

Out[22]= $S^{k^*}{}_{j^*} \kappa^{j^*}{}_{r^*}$

Out[23]= $S^{k^*}{}_{r^*}$

We can even mix two different versions of Kronecker in an expression and then absorb them selectively...

```
In[24]:= δud[m, i] κud[n, j] Suu[i, j] // ToFlavor[red]
         % // KroneckerAbsorb[δ]
         % // KroneckerAbsorb[κ]
```

Out[24]= $S^{ij} \delta^m{}_i \kappa^n{}_j$

Out[25]= $S^{mj} \kappa^n{}_j$

Out[26]= S^{mn}

Kronecker absorption with derivative expressions.

```
In[27]:= δud[j, r] {PartialD[Su[k], j], CovariantD[Su[k], j]} // ToFlavor[red]
         % // KroneckerAbsorb[δ]
```

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

Out[28]= $\{S^{k}{}_{,r},\ S^{k}{}_{;r}\}$

```
In[29]:= TotalD[xu[j], t] δud[r, j]
         % // KroneckerAbsorb[δ]
```

Out[29]= $\delta^{r}{}_{j}\,\frac{dx^{j}}{dt}$

Out[30]= $\frac{dx^{r}}{dt}$

```
In[31]:= δdu[i, ν] δud[k, μ] PartialD[{x, δ, g, Γ}][Sud[i, j], xu[k]]
         % // KroneckerAbsorb[δ]
```

Out[31]= $\delta^{k}{}_{\mu}\,\delta_{i}{}^{\nu}\,\frac{\partial S^{i}{}_{j}}{\partial x^{k}}$

Out[32]= $\frac{\partial S^{\nu}{}_{j}}{\partial x^{\mu}}$

And with dot products and CircleTimes Expressions

```
In[33]:= δud[r, i] {a xu[j].xu[i], (Sin[θ] + Cos[θ]) xu[j] ⊗ xu[i]}
         % // KroneckerAbsorb[δ]
```

Out[33]= $\{a\,x^{j}.x^{i}\,\delta^{r}{}_{i},\ x^{j}\otimes x^{i}\,(\mathrm{Cos}[\theta]+\mathrm{Sin}[\theta])\,\delta^{r}{}_{i}\}$

Out[34]= $\{a\,x^{j}.x^{r},\ x^{j}\otimes x^{r}\,(\mathrm{Cos}[\theta]+\mathrm{Sin}[\theta])\}$

If we expand before fully simplifying, we have to give the Kronecker values. This is a case where we might want to use Values instead of Rules. But students might want to see the steps. With Rules...

```
In[35]:= SetTensorValueRules[κud[i, j] // ToFlavor[red], IdentityMatrix[NDim]]
```

```
In[36]:= δud[m, i] κud[n, j] Suu[i, j] // ToFlavor[red]
         % // KroneckerAbsorb[δ]
         % // EinsteinSum[] // EinsteinArray[] // MatrixForm
         % /. TensorValueRules[κ] // MatrixForm
```

Out[36]= $S^{ij}\,\delta^{m}{}_{i}\,\kappa^{n}{}_{j}$

Out[37]= $S^{mj}\,\kappa^{n}{}_{j}$

Out[38]//MatrixForm=

$$\begin{pmatrix} S^{11}\kappa^{1}{}_{1}+S^{12}\kappa^{1}{}_{2}+S^{13}\kappa^{1}{}_{3} & S^{11}\kappa^{2}{}_{1}+S^{12}\kappa^{2}{}_{2}+S^{13}\kappa^{2}{}_{3} & S^{11}\kappa^{3}{}_{1}+S^{12}\kappa^{3}{}_{2}+S^{13}\kappa^{3}{}_{3} \\ S^{21}\kappa^{1}{}_{1}+S^{22}\kappa^{1}{}_{2}+S^{23}\kappa^{1}{}_{3} & S^{21}\kappa^{2}{}_{1}+S^{22}\kappa^{2}{}_{2}+S^{23}\kappa^{2}{}_{3} & S^{21}\kappa^{3}{}_{1}+S^{22}\kappa^{3}{}_{2}+S^{23}\kappa^{3}{}_{3} \\ S^{31}\kappa^{1}{}_{1}+S^{32}\kappa^{1}{}_{2}+S^{33}\kappa^{1}{}_{3} & S^{31}\kappa^{2}{}_{1}+S^{32}\kappa^{2}{}_{2}+S^{33}\kappa^{2}{}_{3} & S^{31}\kappa^{3}{}_{1}+S^{32}\kappa^{3}{}_{2}+S^{33}\kappa^{3}{}_{3} \end{pmatrix}$$

Out[39]//MatrixForm=

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

With Values...

```
In[40]:= SetTensorValues[κud[i, j] // ToFlavor[red], IdentityMatrix[NDim]]

In[41]:= δud[m, i] κud[n, j] Suu[i, j] // ToFlavor[red]
         % // KroneckerAbsorb[δ]
         % // EinsteinSum[] // EinsteinArray[] // MatrixForm
```

Out[41]= $S^{ij}\,\delta^{m}{}_{i}\,\kappa^{n}{}_{j}$

Out[42]= $S^{mj}\,\kappa^{n}{}_{j}$

Out[43]//MatrixForm=

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

We can also use a trick of MetricSimplify. The up/down forms of the metric are just Kroneckers and MetricSimplify knows how to evaluate on the base indices. So in the following we just make believe that κ is a metric tensor. We don't have to set values.

```
In[44]:= ClearTensorValues[κud[i, j] // ToFlavor[red]]

In[45]:= δud[m, i] κud[n, j] Suu[i, j] // ToFlavor[red]
         % // KroneckerAbsorb[δ]
         % // EinsteinSum[] // EinsteinArray[] // MatrixForm
         % // MetricSimplify[κ] // MatrixForm
```

Out[45]= $S^{ij}\,\delta^{m}{}_{i}\,\kappa^{n}{}_{j}$

Out[46]= $S^{mj}\,\kappa^{n}{}_{j}$

Out[47]//MatrixForm=

$$\begin{pmatrix} S^{11}\kappa^{1}{}_{1}+S^{12}\kappa^{1}{}_{2}+S^{13}\kappa^{1}{}_{3} & S^{11}\kappa^{2}{}_{1}+S^{12}\kappa^{2}{}_{2}+S^{13}\kappa^{2}{}_{3} & S^{11}\kappa^{3}{}_{1}+S^{12}\kappa^{3}{}_{2}+S^{13}\kappa^{3}{}_{3} \\ S^{21}\kappa^{1}{}_{1}+S^{22}\kappa^{1}{}_{2}+S^{23}\kappa^{1}{}_{3} & S^{21}\kappa^{2}{}_{1}+S^{22}\kappa^{2}{}_{2}+S^{23}\kappa^{2}{}_{3} & S^{21}\kappa^{3}{}_{1}+S^{22}\kappa^{3}{}_{2}+S^{23}\kappa^{3}{}_{3} \\ S^{31}\kappa^{1}{}_{1}+S^{32}\kappa^{1}{}_{2}+S^{33}\kappa^{1}{}_{3} & S^{31}\kappa^{2}{}_{1}+S^{32}\kappa^{2}{}_{2}+S^{33}\kappa^{2}{}_{3} & S^{31}\kappa^{3}{}_{1}+S^{32}\kappa^{3}{}_{2}+S^{33}\kappa^{3}{}_{3} \end{pmatrix}$$

Out[48]//MatrixForm=

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

A more complicated expression.

```
In[49]:= Exp[1/2 ((xu[a] + yu[a]) Suu[i, s] δud[r, i] + (xu[a] + yu[a]) Suu[s, j] δud[r, j])]
         % // KroneckerAbsorb[δ]
```

Out[49]= $e^{\frac{1}{2}\left(S^{is}\,(x^{a}+y^{a})\,\delta^{r}{}_{i}+S^{sj}\,(x^{a}+y^{a})\,\delta^{r}{}_{j}\right)}$

Out[50]= $e^{\frac{1}{2}\left(S^{rs}\,(x^{a}+y^{a})+S^{sr}\,(x^{a}+y^{a})\right)}$

Restore the initial state...

```
In[51]:= ClearTensorValues[κud[i, j] // ToFlavor[red]]
         ClearTensorShortcuts[{x, 1}, {{δ, κ, S}, 2}]
```

```
In[53]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

KroneckerContract

■ `KroneckerContract[δ][expr]` will contract all tensors with label δ, assumed to be generalized Kroneckers, to give a numerical factor times a Kronecker on the free indices.

In Tensorial all Kronecker symbols must have one Void in each slot just like all other indexed objects. Many texts use indices in both the up and down positions, taking advantage of the fact that Kroneckers must be even order with equal number of up and down indices.

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

See also: `FullKroneckerExpand`, `PartialKroneckerExpand`, `KroneckerAbsorb`, `KroneckerEvaluate`.

Examples

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

Save the index flavor settings.

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

In[5]:=
```
DefineTensorShortcuts[δ, #] & /@ {2, 4, 6};
DefineTensorShortcuts[κ, 6]
```

The following examples contract on 3, 2, 1 and 0 indices. They might be considered the "standard form" for generalized Kroneckers - all the up indices in the first slots and all the down indices in the last slots.

In[7]:=
```
δuuuddd[i, j, k, i, j, k]
% // KroneckerContract[δ]
```

Out[7]= $\delta^{i\,j\,k}{}_{i\,j\,k}$

Out[8]= 6

In[9]:=
```
δuuuddd[i, j, k, i, j, t]
% // KroneckerContract[δ]
```

Out[9]= $\delta^{i\,j\,k}{}_{i\,j\,t}$

Out[10]= $2\,\delta^{k}{}_{t}$

In[11]:=
```
δuuuddd[i, j, k, i, s, t]
% // KroneckerContract[δ]
```

Out[11]= $\delta^{i\,j\,k}{}_{i\,s\,t}$

Out[12]= $\delta^{j\,k}{}_{s\,t}$

```
In[13]:= δuuuddd[i, j, k, r, s, t]
         % // KroneckerContract[δ]
```

Out[13]= $\delta^{i\,j\,k}{}_{r\,s\,t}$

Out[14]= $\delta^{i\,j\,k}{}_{r\,s\,t}$

A label other than δ can be used to represent the Kronecker.

```
In[15]:= κuuuddd[i, j, k, i, s, t]
         % // KroneckerContract[κ]
```

Out[15]= $\kappa^{i\,j\,k}{}_{i\,s\,t}$

Out[16]= $\kappa^{j\,k}{}_{s\,t}$

The following are nonstandard form Kroneckers. But they can always be converted to standard form by a series of symmetrical up/down interchanges always pushing up indices to the left.

```
In[17]:= δduudud[i, i, s, s, k, k] // ToFlavor[red]
         % // KroneckerContract[δ]
```

Out[17]= $\delta_{i}{}^{i\,s}{}_{s}{}^{k}{}_{k}$

Out[18]= 6

```
In[19]:= δduudud[i, i, s, s, k, t] // ToFlavor[red]
         % // KroneckerContract[δ]
```

Out[19]= $\delta_{i}{}^{i\,s}{}_{s}{}^{k}{}_{t}$

Out[20]= $2\,\delta^{k}{}_{t}$

```
In[21]:= δduudud[i, i, j, s, k, t] // ToFlavor[red]
         % // KroneckerContract[δ]
```

Out[21]= $\delta_{i}{}^{i\,j}{}_{s}{}^{k}{}_{t}$

Out[22]= $\delta^{j\,k}{}_{s\,t}$

```
In[23]:= δduudud[r, i, j, s, k, t] // ToFlavor[red]
         % // KroneckerContract[δ]
```

Out[23]= $\delta_{r}{}^{i\,j}{}_{s}{}^{k}{}_{t}$

Out[24]= $\delta_{r}{}^{i\,j}{}_{s}{}^{k}{}_{t}$

The routine correctly handles permutations (up order different than down order) of the dummy indices.

```
In[25]:= δuuuddd[i, j, k, j, i, k] // ToFlavor[red]
         % // KroneckerContract[δ]
```

Out[25]= $\delta^{i\,j\,k}{}_{j\,i\,k}$

Out[26]= -6

```
In[27]:= δuuuddd[i, j, k, k, i, j] // ToFlavor[red]
         % // KroneckerContract[δ]
```

Out[27]= $\delta^{ijk}{}_{kij}$

Out[28]= 6

```
In[29]:= δuuuddd[i, j, k, j, i, t] // ToFlavor[red]
         % // KroneckerContract[δ]
```

Out[29]= $\delta^{ijk}{}_{jit}$

Out[30]= $-2\,\delta^{k}{}_{t}$

```
In[31]:= δduudud[s, i, s, i, k, k] // ToFlavor[red]
         % // KroneckerContract[δ]
```

Out[31]= $\delta_{s}{}^{is}{}_{i}{}^{k}{}_{k}$

Out[32]= -6

The routine checks for unbalanced indices.

```
In[33]:= δduuuud[r, i, j, s, k, t]
         % // KroneckerContract[δ]
```

Out[33]= $\delta_{r}{}^{ijsk}{}_{t}$

KroneckerContract::indices :
Kronecker tensor $\delta_{r}{}^{ijsk}{}_{t}$ does not have equal up and down indices.

Out[34]= $Aborted

Restore the initial values...

```
In[35]:= ClearTensorShortcuts[δ, #] & /@ {2, 4, 6};
         ClearTensorShortcuts[κ, 6]
```

```
In[37]:= ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

KroneckerEvaluate

- `KroneckerEvaluate[δ][expr]` will evaluate Kronecker symbols in expr whose indices are entirely single flavor base indices.

Generalized Kroneckers are also evaluated.

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

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

Examples

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

Save the settings and declare base indices and flavors.

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]
        DeclareBaseIndices[{1, 2, 3}, {red, {1, 2, 3}}]
        DefineTensorShortcuts[{{δ, κ}, 2}, {δ, 4}, {δ, 6}]
```

The following illustrates various evaluations. The δ tensor is evaluated only if it uses base indices, all of the same flavor, and there are an equal number of up and down indices.

```
In[8]:= {δuu[1, 2], δuudd[1, red@2, 2, red@1], δuudd[a, b, c, d], δud[1, 1],
          δud[1, 2], δud[red@1, red@1], δud[0, 0], δuuuddd[1, 2, 3, 2, 3, 1],
          δuuuddd[1, 2, 3, 2, 1, 3], δududud[1, 1, 2, 2, 3, 3], δududud[2, 1, 1, 2, 3, 3]};
        Thread[% → (% // KroneckerEvaluate[δ])] // TableForm
```

Out[9]//TableForm=

$\delta^{1\,2} \to \delta^{1\,2}$

$\delta^{1\,2}{}_{2\,1} \to \delta^{1\,2}{}_{2\,1}$

$\delta^{a\,b}{}_{c\,d} \to \delta^{a\,b}{}_{c\,d}$

$\delta^{1}{}_{1} \to 1$

$\delta^{1}{}_{2} \to 0$

$\delta^{1}{}_{1} \to 1$

$\delta^{0}{}_{0} \to \delta^{0}{}_{0}$

$\delta^{1\,2\,3}{}_{2\,3\,1} \to 1$

$\delta^{1\,2\,3}{}_{2\,1\,3} \to -1$

$\delta^{1}{}_{1}{}^{2}{}_{2}{}^{3}{}_{3} \to 1$

$\delta^{2}{}_{1}{}^{1}{}_{2}{}^{3}{}_{3} \to -1$

The following is the evaluation of the ordinary Kronecker.

```
In[10]:= δud[a, b] // ToArrayValues[] // MatrixForm
         % // KroneckerEvaluate[δ] // MatrixForm
```

Out[10]//MatrixForm=

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

Out[11]//MatrixForm=

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

The following is the evaluation of an ordinary Kronecker in the red flavor using κ as the Kronecker symbol.

```
In[12]:= κud[a, b] // ToFlavor[red] // ToArrayValues[] // MatrixForm
         % // KroneckerEvaluate[κ] // MatrixForm
```

Out[12]//MatrixForm=

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

Out[13]//MatrixForm=

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

The following is an evaluation of the second order Kronecker.

```
In[14]:= δuudd[a, b, c, d] // ToArrayValues[] // MatrixForm
         % // KroneckerEvaluate[δ] // MatrixForm // MatrixForm
```

Out[14]//MatrixForm=

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

Out[15]//MatrixForm=

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

```
In[16]:= δuuuddd[a, b, c, d, e, f] // ToArrayValues[] // MatrixForm
         % // KroneckerEvaluate[δ] // MatrixForm // MatrixForm
```

```
Out[16]//MatrixForm=
```

$$\left(\begin{matrix}
\left(\begin{matrix}
\begin{pmatrix} \delta^{111}{}_{111} & \delta^{111}{}_{112} & \delta^{111}{}_{113} \\ \delta^{111}{}_{121} & \delta^{111}{}_{122} & \delta^{111}{}_{123} \\ \delta^{111}{}_{131} & \delta^{111}{}_{132} & \delta^{111}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{111}{}_{211} & \delta^{111}{}_{212} & \delta^{111}{}_{213} \\ \delta^{111}{}_{221} & \delta^{111}{}_{222} & \delta^{111}{}_{223} \\ \delta^{111}{}_{231} & \delta^{111}{}_{232} & \delta^{111}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{111}{}_{311} & \delta^{111}{}_{312} & \delta^{111} \\ \delta^{111}{}_{321} & \delta^{111}{}_{322} & \delta^{111} \\ \delta^{111}{}_{331} & \delta^{111}{}_{332} & \delta^{111} \end{matrix}\right. \\
\begin{pmatrix} \delta^{112}{}_{111} & \delta^{112}{}_{112} & \delta^{112}{}_{113} \\ \delta^{112}{}_{121} & \delta^{112}{}_{122} & \delta^{112}{}_{123} \\ \delta^{112}{}_{131} & \delta^{112}{}_{132} & \delta^{112}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{112}{}_{211} & \delta^{112}{}_{212} & \delta^{112}{}_{213} \\ \delta^{112}{}_{221} & \delta^{112}{}_{222} & \delta^{112}{}_{223} \\ \delta^{112}{}_{231} & \delta^{112}{}_{232} & \delta^{112}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{112}{}_{311} & \delta^{112}{}_{312} & \delta^{112} \\ \delta^{112}{}_{321} & \delta^{112}{}_{322} & \delta^{112} \\ \delta^{112}{}_{331} & \delta^{112}{}_{332} & \delta^{112} \end{matrix}\right. \\
\begin{pmatrix} \delta^{113}{}_{111} & \delta^{113}{}_{112} & \delta^{113}{}_{113} \\ \delta^{113}{}_{121} & \delta^{113}{}_{122} & \delta^{113}{}_{123} \\ \delta^{113}{}_{131} & \delta^{113}{}_{132} & \delta^{113}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{113}{}_{211} & \delta^{113}{}_{212} & \delta^{113}{}_{213} \\ \delta^{113}{}_{221} & \delta^{113}{}_{222} & \delta^{113}{}_{223} \\ \delta^{113}{}_{231} & \delta^{113}{}_{232} & \delta^{113}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{113}{}_{311} & \delta^{113}{}_{312} & \delta^{113} \\ \delta^{113}{}_{321} & \delta^{113}{}_{322} & \delta^{113} \\ \delta^{113}{}_{331} & \delta^{113}{}_{332} & \delta^{113} \end{matrix}\right.
\end{matrix}\right. \\
\left(\begin{matrix}
\begin{pmatrix} \delta^{211}{}_{111} & \delta^{211}{}_{112} & \delta^{211}{}_{113} \\ \delta^{211}{}_{121} & \delta^{211}{}_{122} & \delta^{211}{}_{123} \\ \delta^{211}{}_{131} & \delta^{211}{}_{132} & \delta^{211}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{211}{}_{211} & \delta^{211}{}_{212} & \delta^{211}{}_{213} \\ \delta^{211}{}_{221} & \delta^{211}{}_{222} & \delta^{211}{}_{223} \\ \delta^{211}{}_{231} & \delta^{211}{}_{232} & \delta^{211}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{211}{}_{311} & \delta^{211}{}_{312} & \delta^{211} \\ \delta^{211}{}_{321} & \delta^{211}{}_{322} & \delta^{211} \\ \delta^{211}{}_{331} & \delta^{211}{}_{332} & \delta^{211} \end{matrix}\right. \\
\begin{pmatrix} \delta^{212}{}_{111} & \delta^{212}{}_{112} & \delta^{212}{}_{113} \\ \delta^{212}{}_{121} & \delta^{212}{}_{122} & \delta^{212}{}_{123} \\ \delta^{212}{}_{131} & \delta^{212}{}_{132} & \delta^{212}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{212}{}_{211} & \delta^{212}{}_{212} & \delta^{212}{}_{213} \\ \delta^{212}{}_{221} & \delta^{212}{}_{222} & \delta^{212}{}_{223} \\ \delta^{212}{}_{231} & \delta^{212}{}_{232} & \delta^{212}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{212}{}_{311} & \delta^{212}{}_{312} & \delta^{212} \\ \delta^{212}{}_{321} & \delta^{212}{}_{322} & \delta^{212} \\ \delta^{212}{}_{331} & \delta^{212}{}_{332} & \delta^{212} \end{matrix}\right. \\
\begin{pmatrix} \delta^{213}{}_{111} & \delta^{213}{}_{112} & \delta^{213}{}_{113} \\ \delta^{213}{}_{121} & \delta^{213}{}_{122} & \delta^{213}{}_{123} \\ \delta^{213}{}_{131} & \delta^{213}{}_{132} & \delta^{213}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{213}{}_{211} & \delta^{213}{}_{212} & \delta^{213}{}_{213} \\ \delta^{213}{}_{221} & \delta^{213}{}_{222} & \delta^{213}{}_{223} \\ \delta^{213}{}_{231} & \delta^{213}{}_{232} & \delta^{213}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{213}{}_{311} & \delta^{213}{}_{312} & \delta^{213} \\ \delta^{213}{}_{321} & \delta^{213}{}_{322} & \delta^{213} \\ \delta^{213}{}_{331} & \delta^{213}{}_{332} & \delta^{213} \end{matrix}\right.
\end{matrix}\right. \\
\left(\begin{matrix}
\begin{pmatrix} \delta^{311}{}_{111} & \delta^{311}{}_{112} & \delta^{311}{}_{113} \\ \delta^{311}{}_{121} & \delta^{311}{}_{122} & \delta^{311}{}_{123} \\ \delta^{311}{}_{131} & \delta^{311}{}_{132} & \delta^{311}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{311}{}_{211} & \delta^{311}{}_{212} & \delta^{311}{}_{213} \\ \delta^{311}{}_{221} & \delta^{311}{}_{222} & \delta^{311}{}_{223} \\ \delta^{311}{}_{231} & \delta^{311}{}_{232} & \delta^{311}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{311}{}_{311} & \delta^{311}{}_{312} & \delta^{311} \\ \delta^{311}{}_{321} & \delta^{311}{}_{322} & \delta^{311} \\ \delta^{311}{}_{331} & \delta^{311}{}_{332} & \delta^{311} \end{matrix}\right. \\
\begin{pmatrix} \delta^{312}{}_{111} & \delta^{312}{}_{112} & \delta^{312}{}_{113} \\ \delta^{312}{}_{121} & \delta^{312}{}_{122} & \delta^{312}{}_{123} \\ \delta^{312}{}_{131} & \delta^{312}{}_{132} & \delta^{312}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{312}{}_{211} & \delta^{312}{}_{212} & \delta^{312}{}_{213} \\ \delta^{312}{}_{221} & \delta^{312}{}_{222} & \delta^{312}{}_{223} \\ \delta^{312}{}_{231} & \delta^{312}{}_{232} & \delta^{312}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{312}{}_{311} & \delta^{312}{}_{312} & \delta^{312} \\ \delta^{312}{}_{321} & \delta^{312}{}_{322} & \delta^{312} \\ \delta^{312}{}_{331} & \delta^{312}{}_{332} & \delta^{312} \end{matrix}\right. \\
\begin{pmatrix} \delta^{313}{}_{111} & \delta^{313}{}_{112} & \delta^{313}{}_{113} \\ \delta^{313}{}_{121} & \delta^{313}{}_{122} & \delta^{313}{}_{123} \\ \delta^{313}{}_{131} & \delta^{313}{}_{132} & \delta^{313}{}_{133} \end{pmatrix} & \begin{pmatrix} \delta^{313}{}_{211} & \delta^{313}{}_{212} & \delta^{313}{}_{213} \\ \delta^{313}{}_{221} & \delta^{313}{}_{222} & \delta^{313}{}_{223} \\ \delta^{313}{}_{231} & \delta^{313}{}_{232} & \delta^{313}{}_{233} \end{pmatrix} & \left(\begin{matrix} \delta^{313}{}_{311} & \delta^{313}{}_{312} & \delta^{313} \\ \delta^{313}{}_{321} & \delta^{313}{}_{322} & \delta^{313} \\ \delta^{313}{}_{331} & \delta^{313}{}_{332} & \delta^{313} \end{matrix}\right.
\end{matrix}\right.
\end{matrix}\right.$$

Out[17]//MatrixForm=

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

Restore the initial state...

In[18]:=
```
ClearTensorShortcuts[{{δ, κ}, 2}, {δ, 4}, {δ, 6}]
```

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

LHSSymbolsToPatterns

■ `LHSSymbolsToPatterns[`*symbollist*`][`*expr*`]` will apply `SymbolsToPatterns` to the first part of expr.

This is a method of changing derived equations into general rules that can be used to perform substitutions in further derivations.

See also: `SymbolsToPatterns`.

Examples

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

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

The following equation can be changed into a rule.

```
In[5]:= eqn1 = Tdd[μ, ν] == (ρ + p) ud[μ] ud[ν] - p gdd[μ, ν]
        Print["To make a general rule with p and ρ as parameters and μ,ν as patterns"]
        rule1[p_, ρ_] = Rule @@ eqn1 // LHSSymbolsToPatterns[{μ, ν}]
```

Out[5]= $T_{\mu\nu} == -p\, g_{\mu\nu} + (p + \rho)\, u_\mu\, u_\nu$

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

Out[7]= $T_{\mu\_\,\nu\_} \to -p\, g_{\mu\nu} + (p + \rho)\, u_\mu\, u_\nu$

This can then be used to substitute in equations.

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

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

Out[9]= $G_{\alpha\beta} == 8\,\pi\, (-p\, g_{\alpha\beta} + (p + \rho)\, u_\alpha\, u_\beta)$

Restore state

```
In[10]:= ClearTensorShortcuts[{u, 1}, {{g, T, G}, 2}]
```

```
In[11]:= DeclareBaseIndices @@ oldindices
         Clear[eqn1, rule1]
```

LieD

- `LieD[`*tensor*, *V*`]` represents the Lie derivative of the tensor with respect to the vector field `V`.
- `LieD[`*tensor*, {*U*, *V*, ...}`]` represents the Lie derivative with respect to the list of vector fields.

The Lie derivative is ambiguous until it is expanded to partial derivatives with `ExpandLieD`, which provides the coordinate positions.

U, V, etc. are tensor labels.

LieD is by default left unformatted, which means it displays as `LieD`. It can be changed to common textbook form using SetLieDisplay and SetDerivativeSymbols.

See also: `ExpandLieD`, `SetLieDisplay`, `SetDerivativeSymbols`, `AbsoluteD`, `CovariantD`, `PartialD`, `TotalD`.

Examples

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

Save the settings.

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

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

LieD is by default left unformatted.

```
In[7]:= Td[i]
        LieD[%, V]
```

Out[7]= T_i

Out[8]= LieD[T_i, V]

The display can be set to common textbook style using the `SetLieDisplay` command.

```
In[9]:= SetLieDisplay["LieMode"]
```

```
In[10]:= Td[i]
         LieD[%, V]
```

Out[10]= T_i

Out[11]= $£_V T_i$

LieD obeys the usual derivative rules. A symbol is considered constant.

```
In[12]:= a Su[i] + 3 Tu[i]
         LieD[%, V]
```

Out[12]= $a\,S^{i} + 3\,T^{i}$

Out[13]= $a\,\pounds_{V}S^{i} + 3\,\pounds_{V}T^{i}$

```
In[14]:= a Tu[i] Wud[j, i]
         LieD[%, V]
```

Out[14]= $a\,T^{i}\,W^{j}{}_{i}$

Out[15]= $a\,\left(\pounds_{V}W^{j}{}_{i}\,T^{i} + \pounds_{V}T^{i}\,W^{j}{}_{i}\right)$

The Lie derivative of a scalar is also defined.

```
In[16]:= a Tensor[ϕ]
         LieD[%, V]
```

Out[16]= $a\,\phi$

Out[17]= $a\,\pounds_{V}\phi$

Higher order derivatives are supported.

```
In[18]:= Tensor[ϕ] Tensor[ψ]
         LieD[%, {V, V}]
```

Out[18]= $\phi\,\psi$

Out[19]= $2\,\pounds_{V}\phi\,\pounds_{V}\psi + \pounds_{V\,V}\psi\,\phi + \pounds_{V\,V}\phi\,\psi$

Nothing special has to be done for flavored expressions.

```
In[20]:= Tensor[ϕ] Wud[i, j] // ToFlavor[red]
         LieD[%, {V, V}]
```

Out[20]= $\phi\,W^{i}{}_{j}$

Out[21]= $2\,\pounds_{V}\phi\,\pounds_{V}W^{i}{}_{j} + \pounds_{V\,V}W^{i}{}_{j}\,\phi + \pounds_{V\,V}\phi\,W^{i}{}_{j}$

The display is reset to an unformatted form with the command

```
In[22]:= SetLieDisplay["PlainMode"]
```

```
In[23]:= Tensor[ϕ] Wud[i, j] // ToFlavor[red]
         LieD[%, {V, V}]
```

Out[23]= $\phi\,W^{i}{}_{j}$

Out[24]= $2\,\mathrm{LieD}[\phi, V]\,\mathrm{LieD}\left[W^{i}{}_{j}, V\right] + \mathrm{LieD}\left[W^{i}{}_{j}, \{V, V\}\right]\,\phi + \mathrm{LieD}[\phi, \{V, V\}]\,W^{i}{}_{j}$

Restore the settings.

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

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

LinearBreakout

- `LinearBreakout[`*f1, f2, ...*`][`*v1, v2, ...*`][expr]` will break out the linear terms of any expressions within expr that have heads matching the patterns fi over variables matching the patterns vj.

Since tensor calculus is a theory of multilinear functions, LinearBreakout is a very useful routine for manipulating expressions.

See also: `PushOnto`, `BasisDotProductRules`, `EvaluateSlots`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
        DefineTensorShortcuts[{{u, v, w, g}, 1}, {{T, g}, 2}]
```

The following is about the simplest example. The functions f and g are linear in x and y.

```
In[3]:= f[a x + b y] + g[c x + d y]
        % // LinearBreakout[f, g][x, y]
```

```
Out[3]= f[a x + b y] + g[c x + d y]
```

```
Out[4]= a f[x] + b f[y] + c g[x] + d g[y]
```

The following breaks out the dot product of two vectors.

```
In[5]:= u.v
        % /. {u → uu[i] gd[i], v → vu[j] gd[j]}
        % // LinearBreakout[Dot][gd[_]]
```

```
Out[5]= u.v
```

Out[6]= $(g_i\, u^i).(g_j\, v^j)$

Out[7]= $g_i.g_j\, u^i\, v^j$

In the following we want to breakout a triple product on Dot and Cross, but we have to breakout the Dot on Cross patterns also.

```
In[8]:= u×v.w
        % /. {u → uu[i] gd[i], v → vu[j] gd[j], w → wu[k] gd[k]}
        % // LinearBreakout[Dot, Cross][Cross[__], gd[_]]
```

```
Out[8]= u×v.w
```

Out[9]= $(g_i\, u^i) \times (g_j\, v^j).(g_k\, w^k)$

Out[10]= $g_i \times g_j.g_k\, u^i\, v^j\, w^k$

If T is a second order tensor, then we can breakout a slot expression on the basis vectors.

```
In[11]:= T[u, v]
         % /. {u → uu[i] gd[i], v → vu[j] gd[j]}
         % // LinearBreakout[T][gd[_]]
         % /. T[gd[i_], gd[j_]] → Tdd[i, j]
```

Out[11]= $T[u, v]$

Out[12]= $T\left[g_i u^i, g_j v^j\right]$

Out[13]= $T\left[g_i, g_j\right] u^i v^j$

Out[14]= $T_{i\,j} u^i v^j$

In the following we create a dyad tensor (direct product) from u and v and then evaluate it on w. (We are only evaluating on the second slot.)

```
In[15]:= (u⊗v)[, w]
         % /. {u → uu[i] gd[i], v → vu[j] gd[j], w → wu[k] gd[k]}
         % // LinearBreakout[CircleTimes][gd[_]]
         % // PushOnto[{CircleTimes[__]}]
         % // EvaluateSlots[g, δ]
```

Out[15]= $(u \otimes v)[\text{Null}, w]$

Out[16]= $\left((g_i u^i) \otimes \left(g_j v^j\right)\right)[\text{Null}, g_k w^k]$

Out[17]= $\left(g_i \otimes g_j u^i v^j\right)[\text{Null}, g_k w^k]$

Out[18]= $u^i v^j \left(g_i \otimes g_j\right)[\text{Null}, g_k w^k]$

Out[19]= $g_i u^i v_k w^k$

But EvaluateSlots, itself, actually does the LinearBreakout and PushOnto.

```
In[20]:= (u⊗v)[, w]
         % /. {u → uu[i] gd[i], v → vu[j] gd[j], w → wu[k] gd[k]}
         % // EvaluateSlots[g, δ]
```

Out[20]= $(u \otimes v)[\text{Null}, w]$

Out[21]= $\left((g_i u^i) \otimes \left(g_j v^j\right)\right)[\text{Null}, g_k w^k]$

Out[22]= $g_i u^i v_k w^k$

```
In[23]:= ClearTensorShortcuts[{{u, v, w, g}, 1}, {{T, g}, 2}]
```

LowerIndex

- `LowerIndex[i, j][`*tensor*`]` will lower the upper index i in tensor and rename it j.
- `LowerIndex[i, j][`*expr*`]` will lower the index in all Tensors in expr.

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

The indices i and j should be raw index Symbols.

See also: `RaiseIndex`, `ReplaceIndex`, `ParseTermIndices`, `MetricSimplify`.

Examples

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

Save the old settings.

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

In[4]:= DeclareIndexFlavor[{red, Red}]
        DefineTensorShortcuts[{{S, T}, 2}]

In[6]:= Sud[i, j]
        % // LowerIndex[i, i]
```

Out[6]= $S^{i}{}_{j}$

Out[7]= $S_{i\,j}$

The raw index is used in the routine...

```
In[8]:= Sud[i, j] // ToFlavor[red]
        % // LowerIndex[i, k]
```

Out[8]= $S^{i}{}_{j}$

Out[9]= $S_{k\,j}$

If a flavored index is used, it should be used in both arguments.

```
In[10]:= Sud[i, j] // ToFlavor[red]
         % // LowerIndex[red@i, k]
```

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

Out[11]= $S_{k\,j}$

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

```
In[12]:= Sud[i, j]
         % // LowerIndex[k, j]
```

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

LowerIndex::noindex : k is not an upper index in $S^{i}{}_{j}$

Out[13]= $Aborted

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

```
In[14]:= Suu[i, j] Tuu[a, b]
         % // LowerIndex[a, c]
```

Out[14]= $S^{i\,j}\,T^{a\,b}$

Out[15]= $S^{i\,j}\,T_{c}{}^{b}$

Restore the initial settings...

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

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