ParseTermIndices

- `ParseTermIndices[`*term*`]` will return the lists of indices: `{dummies, {freeup, freedown}, bad}` for the term.

This routine is primarily used in programming other routines.

Free indices are indices that appear only once in the term. Dummy indices are indices that appear exactly once in an up position and once in a down position. All other indices are bad.

`ParseTermIndices` works on a single term.

`ParseTermIndices` excludes base indices and hence routines that use it will not operate on base indices.

`ParseTermIndices` works on most common types of tensor terms, but not every possible type of term. The `IndexParsingRules` command can be used to set rules so that unanticipated forms of tensor terms can also be parsed.

See also: `IndexParsingRules`, `SimplifyTensorSum`, `EinsteinSum`, `EinsteinArray`, `ExtractFreeIndices`.

Examples

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

Save the old settings.

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

In[7]:= DefineTensorShortcuts[{{x, T}, 1}, {{T, S, η}, 2}, {{T, S}, 3}]
        labs = {x, δ, g, Γ};
```

An up index....

```
In[9]:= xu[i]
        % // ParseTermIndices

Out[9]= x^i

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

A dummy index...

```
In[11]:= xu[j] xd[j]
         % // ParseTermIndices

Out[11]= x^j x_j

Out[12]= {{j}, {{}, {}}, {}}
```

Up, down and dummy indices...

```
In[13]:= Tudd[i, j, k] xu[k]
         % // ParseTermIndices
```

```
Out[13]= T^i_{j k} x^k
```

```
Out[14]= {{k}, {{i}, {j}}, {}}
```

Bad indices....

```
In[15]:= xu[k] xu[k]
         % // ParseTermIndices
```

```
Out[15]= (x^k)^2
```

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

```
In[17]:= Suud[j, i, j] xu[j]
         % // ParseTermIndices
```

```
Out[17]= S^{j i}_j x^j
```

```
Out[18]= {{}, {{i}, {}}, {j}}
```

The flavor is retained.

```
In[19]:= 2 CovariantD[Tud[i, j], k] // ToFlavor[red]
         % // ParseTermIndices
```

```
Out[19]= 2 T^i_{j;k}
```

```
Out[20]= {{}, {{i}, {j, k}}, {}}
```

It works on dot products and CircleTimes products.

```
In[21]:= xu[i].xu[j]
         % // ParseTermIndices
```

```
Out[21]= x^i.x^j
```

```
Out[22]= {{}, {{i, j}, {}}, {}}
```

```
In[23]:= xu[i]⊗Tddu[i, j, k] // ToFlavor[rocket]
         % // ParseTermIndices
```

```
Out[23]= x^{i*} ⊗ T_{i* j*}^{k*}
```

```
Out[24]= {{i*}, {{k*}, {j*}}, {}}
```

Additional types of expressions may be added by using `IndexParsingRules`.

Derivative quantities.

```
In[25]:= {PartialD[labs][Tu[i], xu[j]], PartialD[labs][Tu[i], {xu[i], xd[k], Tensor[ϕ]}],
          TotalD[Tu[a], u], AbsoluteD[Tud[a, b], u], CovariantD[Tdd[a, b], c],
          PartialD[labs][Tensor[s], {xu[i], xd[k], Tensor[ϕ]}], LieD[Td[i], V]}
         ParseTermIndices /@ % // MatrixForm
```

$$\text{Out[25]=} \left\{\frac{\partial T^i}{\partial x^j}, \frac{\partial^3 T^i}{\partial x^i \partial x_k \partial \phi}, \frac{d T^a}{d u}, \frac{D T^a{}_b}{d u}, T_{a b;c}, \frac{\partial^3 s}{\partial x^i \partial x_k \partial \phi}, \text{LieD}[T_i, V]\right\}$$

```
Out[26]//MatrixForm=
{{}    {{i}, {j}}       {}}
{{i}   {{k}, {}}        {}}
{{}    {{a}, {}}        {}}
{{}    {{a}, {b}}       {}}
{{}    {{}, {a, b, c}}  {}}
{{}    {{k}, {i}}       {}}
{{}    {{}, {i}}        {}}
```

It will work on nested tensors provided the nested tensor contains no sums.

```
In[27]:= PartialD[Tensor[Suu[a, b] Td[b]], {j, k}]
         % // ParseTermIndices
```

$$\text{Out[27]=} \left(S^{a b} T_b\right)_{,j k}$$

```
Out[28]= {{b}, {{a}, {j, k}}, {}}
```

The following tests the case of the same raw index in the dummies and in flavored free indices.

```
In[29]:= Tud[r, red@s] Tud[t, red@r] ηdd[t, r]
         % // ParseTermIndices
```

$$\text{Out[29]=} T^r{}_s T^t{}_r \eta_{t r}$$

```
Out[30]= {{r, t}, {{}, {s, r}}, {}}
```

Tensors can be written as functions with arguments, but any Tensors in the arguments are eliminated.

```
In[31]:= Tdd[a, b][xu[i]]
         % // ParseTermIndices
```

$$\text{Out[31]=} T_{a b}[x^i]$$

```
Out[32]= {{}, {{}, {a, b}}, {}}
```

Restore the initial settings...

```
In[33]:= ClearTensorShortcuts[{{x, T}, 1}, {{T, S}, 2}, {{T, S}, 3}]

In[34]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

PartialArray

- `PartialArray[`*baseindex, newindices*`][`*expr*`]` will expand the free indices of a tensor expression into components that have one or more baseindex indices and an unexpanded remainder. Symbolic indices are replaced with newindices.

The elements are returned as a list.

The newindices list of symbols is used to replace the existing free index symbols. Greek indices might be replaced with Latin indices to indicate a restricted further expansion. But it is up to the user to actually use a restriced expansion when further expansions are performed.

Any flavor must be on the newindices and on the baseindex. Only indices with the baseindex flavor will be partially expanded. This is a change in usage from Version 3.0.

PartialArray is mapped over arrays, equations and sums.

See also: `PartialSum`, `EinsteinSum`, `ArrayExpansion`, `SumExpansion`.

Examples

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

Save the settings and declare base indices and flavors.

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareBaseIndices[{0, 1, 2, 3}]
        DeclareIndexFlavor[{red, Red}, {blue, Blue}];

In[7]:= DefineTensorShortcuts[{{λ}, 1}, {T, 2}, {T, 3}]
```

Let λ be a spacetime vector. The following expands it into its components.

```
In[8]:= λu[μ]
        % // EinsteinArray[]

Out[8]= λ^μ

Out[9]= {λ^0, λ^1, λ^2, λ^3}
```

If instead we want to break λ into its temporal and spatial components we can use

```
In[10]:= λu[μ]
         step1 = % // PartialArray[0, {i}]

Out[10]= λ^μ

Out[11]= {λ^0, λ^i}
```

0 is the index for the temporal part. The Greek index μ was replaced with the Latin i to indicate that the remaining part should have a restricted expansion. We could complete the expansion with

```
In[12]:= MapAt[EinsteinArray[{1, 2, 3}], step1, 2]
```

Out[12]= $\{\lambda^0, \{\lambda^1, \lambda^2, \lambda^3\}\}$

The flavor must be on the replacement index and on the base index.

```
In[13]:= λu[red@μ]
         % // PartialArray[red@0, {red@i}]
```

Out[13]= λ^{μ}

Out[14]= $\{\lambda^0, \lambda^i\}$

The following expands a rank 2 spacetime tensor on the temporal part and then completes the expansion by mapping `EinsteinArray` onto the parts. We have to map `EinsteinArray` because the parts do not have matching free indices.

```
In[15]:= Tuu[μ, ν]
         step1 = % // PartialArray[0, {i, j}]
         EinsteinArray[{1, 2, 3}] /@ %
```

Out[15]= $T^{\mu\nu}$

Out[16]= $\left\{T^{00}, T^{0j}, T^{i0}, T^{ij}\right\}$

Out[17]= $\{T^{00}, \{T^{01}, T^{02}, T^{03}\}, \{T^{10}, T^{20}, T^{30}\},$
$\{\{T^{10}, T^{11}, T^{12}, T^{13}\}, \{T^{20}, T^{21}, T^{22}, T^{23}\}, \{T^{30}, T^{31}, T^{32}, T^{33}\}\}\}$

Successive expansions can be performed on various base indices. The following expands first on the 0 index, then on the 1 index and then on the remaining indices.

```
In[18]:= Tuu[μ, ν] // ToFlavor[red]
         % // PartialArray[red@0, red /@ {i, j}]
         % // PartialArray[red@1, red /@ {i, j}]
         Map[EinsteinArray[{2, 3}], %, {2}]
```

Out[18]= $T^{\mu\nu}$

Out[19]= $\left\{T^{00}, T^{0j}, T^{i0}, T^{ij}\right\}$

Out[20]= $\left\{\{T^{00}\}, \{T^{01}, T^{0i}\}, \{T^{10}, T^{i0}\}, \left\{T^{11}, T^{1j}, T^{i1}, T^{ij}\right\}\right\}$

Out[21]= $\{\{T^{00}\}, \{T^{01}, \{T^{02}, T^{03}\}\}, \{T^{10}, \{T^{20}, T^{30}\}\},$
$\{T^{11}, \{T^{12}, T^{13}\}, \{T^{21}, T^{31}\}, \{\{T^{20}, T^{21}, T^{22}, T^{23}\}, \{T^{30}, T^{31}, T^{32}, T^{33}\}\}\}$

If `PartialArray` is applied to an expression without matching indices an error is generated.

```
In[22]:= Tuu[μ, ν] λd[σ] + Tuud[μ, ν, β] // ToFlavor[red]
         % // PartialArray[red@0, red /@ {i, j, k}]
```

Out[22]= $T^{\mu\nu}{}_{\beta} + T^{\mu\nu}\,\lambda_{\sigma}$

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

Out[23]= $Aborted

The new indices must be sufficient to replace the old free indices after removing any conflicts with dummy indices.

```
In[24]:= Tuud[μ, ν, i] λu[i] // ToFlavor[red]
         % // PartialArray[red@0, red /@ {i, j}]
```

$$\text{Out[24]=}\quad T^{\mu\,\nu}{}_{i}\,\lambda^{i}$$

```
PartialArray::newindices :
 New indices {j}, after removing dummies {i}, are insufficient to replace {μ, ν}
```

```
Out[25]= $Aborted
```

The routine works with multiple base index sets.

```
In[26]:= DeclareBaseIndices[{t, x, y, z}, {red, {t, ρ, θ, ϕ}}, {blue, {ρ, θ, ϕ}}]
```

```
In[27]:= λu[μ]
         step1 = % // PartialArray[t, {i}]
         MapAt[EinsteinArray[{x, y, z}], step1, 2]
```

$$\text{Out[27]=}\quad \lambda^{\mu}$$

$$\text{Out[28]=}\quad \{\lambda^{t},\ \lambda^{i}\}$$

$$\text{Out[29]=}\quad \{\lambda^{t},\ \{\lambda^{x},\ \lambda^{y},\ \lambda^{z}\}\}$$

```
In[30]:= λu[red@μ]
         step1 = % // PartialArray[red@t, {red@i}]
         MapAt[EinsteinArray[{ρ, θ, ϕ}], step1, 2]
```

$$\text{Out[30]=}\quad \lambda^{\mu}$$

$$\text{Out[31]=}\quad \{\lambda^{t},\ \lambda^{i}\}$$

$$\text{Out[32]=}\quad \{\lambda^{t},\ \{\lambda^{\rho},\ \lambda^{\theta},\ \lambda^{\phi}\}\}$$

The following may be a more convenient method. We set blue indices above to encompass only the spatial dimensions. Then by substituting a blue μ for a red μ, the array expansion will automatically be confined to the spatial dimensions. We still had to apply `EinsteinArray` to the individual terms so we would not get a conflict in the free indices.

```
In[33]:= λu[red@μ]
         % // PartialArray[red@t, {blue@μ}]
         EinsteinArray[] /@ %
```

$$\text{Out[33]=}\quad \lambda^{\mu}$$

$$\text{Out[34]=}\quad \{\lambda^{t},\ \lambda^{\mu}\}$$

$$\text{Out[35]=}\quad \{\lambda^{t},\ \{\lambda^{\rho},\ \lambda^{\theta},\ \lambda^{\phi}\}\}$$

Only indices of the baseindex flavor are broken out.

```
In[36]:= Tuu[μ, ν] λd[red@σ]
         % // PartialArray[red@t, {red@i}]
```

$$\text{Out[36]=}\quad T^{\mu\,\nu}\,\lambda_{\sigma}$$

$$\text{Out[37]=}\quad \{T^{\mu\,\nu}\,\lambda_{t},\ T^{\mu\,\nu}\,\lambda_{i}\}$$

```
In[38]:= Tuu[μ, ν] λd[red@σ]
         % // PartialArray[t, {i, j}]
         PartialArray[red@t, {red@i}] /@ %
```

Out[38]= $T^{\mu\,\nu}\,\lambda_{\sigma}$

Out[39]= $\left\{T^{t\,t}\,\lambda_{\sigma},\ T^{t\,j}\,\lambda_{\sigma},\ T^{i\,t}\,\lambda_{\sigma},\ T^{i\,j}\,\lambda_{\sigma}\right\}$

Out[40]= $\left\{\{T^{t\,t}\,\lambda_{t},\ T^{t\,t}\,\lambda_{i}\},\ \left\{T^{t\,j}\,\lambda_{t},\ T^{t\,j}\,\lambda_{i}\right\},\ \{T^{i\,t}\,\lambda_{t},\ T^{i\,t}\,\lambda_{i}\},\ \left\{T^{i\,j}\,\lambda_{t},\ T^{i\,j}\,\lambda_{i}\right\}\right\}$

```
In[41]:= Tuu[μ, ν] λd[red@σ]
         % // PartialArray[t, {i, j}]
         % // PartialArray[red@t, {red@i}]
```

Out[41]= $T^{\mu\,\nu}\,\lambda_{\sigma}$

Out[42]= $\left\{T^{t\,t}\,\lambda_{\sigma},\ T^{t\,j}\,\lambda_{\sigma},\ T^{i\,t}\,\lambda_{\sigma},\ T^{i\,j}\,\lambda_{\sigma}\right\}$

Out[43]= $\left\{\{T^{t\,t}\,\lambda_{t},\ T^{t\,t}\,\lambda_{i}\},\ \left\{T^{t\,j}\,\lambda_{t},\ T^{t\,j}\,\lambda_{i}\right\},\ \{T^{i\,t}\,\lambda_{t},\ T^{i\,t}\,\lambda_{i}\},\ \left\{T^{i\,j}\,\lambda_{t},\ T^{i\,j}\,\lambda_{i}\right\}\right\}$

Restore the initial values...

```
In[44]:= ClearTensorShortcuts[{{λ}, 1}, {T, 2}, {T, 3}]
```

```
In[45]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

PartialD

- `PartialD[`*tensor, i*`]` represents the partial derivative of the tensor with respect to the coordinate of index i.
- `PartialD[`*tensor,* {*i, j,* ...}`]` represents higher order derivatives.
- `PartialD[{`*x, δ, g, Γ*`}][`*tensor, difvar*`]` expands the partial derivative of the tensor with respect to difvar using a coordinate position x and the Kronecker delta, *δ*.
- `PartialD[{`*x, δ, g, Γ*`}][`*tensor,* {*difvar1, difvar2,* ...}`]` expands the partial derivative with respect to a number of differentiation variables.

A differentiation variable, difvar, can be a symbolic variable or an x coordinate position, `xu[j]` (in shortcut notation).

The partial derivative of a tensor is not itself a tensor. However, it is used along with the Christoffel symbols in calculating the covariant derivative, which is a proper tensor.

Partial derivatives are done in the order of the indices, left to right but the order will not matter on continuous functions.

In the expanded form of `PartialD` the extended list of labels `{x, δ, g, Γ}`, for coordinate, Kronecker, metric and Christoffel labels, is given to be in conformity with similar usage with the other derivatives. Only `x` and *δ* are actually used with PartialD.

When working in a notebook with a constant set of labels one can put `labs = {x, δ, g, Γ}` and then use `PartialD[labs][tensor, var]` in the extended call, with similar usage for the other derivative routines.

See also: `SetPartialDisplay`, `ExpandPartialD`, `NondependentPartialD`, `TotalD`, `CovariantD`, `AbsoluteD`.

Examples

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

Save settings.

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

Define the tensor shortcuts and derivative labels.

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

Without information as to the coordinate, a partial derivative simply shows a comma before the differentiated indices. PartialD is inert when the first argument is a tensor.

```
In[8]:= Td[i]
        PartialD[%, j]
        % // FullForm

Out[8]= T_i

Out[9]= T_{i,j}

  Out[10]//FullForm=
      PartialD[Tensor[T, List[Void], List[i]], j]
```

Higher order partial derivatives are supported.

```
In[11]:= Tu[i]
         PartialD[%, {m, n, p}]

Out[11]= T^i

Out[12]= T^i_{,mnp}
```

The partial derivative of a symbol is zero.

```
In[13]:= PartialD[a, i]

Out[13]= 0
```

`PartialD` obeys the linear and Liebnizian rules of differentiation.

```
In[14]:= a xu[i] + b Tu[i]
         PartialD[%, j]

Out[14]= b T^i + a x^i

Out[15]= b T^i_{,j} + a x^i_{,j}
```

For flavored expressions the intended flavor must also be on the indices.

```
In[16]:= a xu[i] Tu[j] // ToFlavor[red]
         PartialD[%, red@k]

Out[16]= a T^j x^i

Out[17]= a (x^i_{,k} T^j + T^j_{,k} x^i)
```

In the extended form of `PartialD`, using the derivative labels, the coordinate position itself, rather than just the index must be given. The expression is stored internally with an `PartialD[labs]` header, but is formatted as a partial derivative.

```
In[18]:= Td[i]
         PartialD[labs][%, xu[j]]
         % // FullForm
```

Out[18]= T_i

Out[19]= $\frac{\partial T_i}{\partial x^j}$

```
Out[20]//FullForm=
      PartialD[List[x, \[Delta], g, \[CapitalGamma]]][
       Tensor[T, List[Void], List[i]], Tensor[x, List[j], List[Void]]]
```

Derivatives of the coordinate give the Kronecker delta.

```
In[21]:= xu[i]
         PartialD[labs][%, xu[j]]
```

Out[21]= x^i

Out[22]= $\delta^i{}_j$

To obtain a Kronecker the flavor of the indices must match.

```
In[23]:= xu[i]
         PartialD[labs][%, xu[red@j]]
```

Out[23]= x^i

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

Partial derivatives may also be taken with respect to symbolic variables.

```
In[25]:= Tu[i]
         PartialD[labs][%, v]
```

Out[25]= T^i

Out[26]= $\frac{\partial T^i}{\partial v}$

Partial derivatives may also be taken with respect to a mixture of coordinate positions and variables.

```
In[27]:= Tu[i]
         PartialD[labs][%, {xu[j], xu[k], v}]
```

Out[27]= T^i

Out[28]= $\frac{\partial^3 T^i}{\partial x^j \partial x^k \partial v}$

Derivatives may be taken sequentially.

```
In[29]:= FoldList[PartialD[labs], Sud[m, n] // ToFlavor[red],
          {xu[i], xu[j], xu[k]} // ToFlavor[red]]
```

Out[29]= $\left\{S^m{}_n, \frac{\partial S^m{}_n}{\partial x^i}, \frac{\partial^2 S^m{}_n}{\partial x^j \partial x^i}, \frac{\partial^3 S^m{}_n}{\partial x^k \partial x^j \partial x^i}\right\}$

Here we calculate the transformation matrix between Cartesian (red) and polar (plain) coordinate positions in the plane.

```
In[30]:= DeclareBaseIndices[{1, 2}]
         SetTensorValueRules[xu[i], {r, ϕ}]
         SetTensorValueRules[xu[red@i], {r Cos[ϕ], r Sin[ϕ]}]
```

```
In[33]:= xu[red@i]
         PartialD[labs][%, xu[j]]
         % // EinsteinArray[] // MatrixForm
         % /. TensorValueRules[x] // MatrixForm
         % /. {Cos[ϕ] → x/√(x^2 + y^2), Sin[ϕ] → y/√(x^2 + y^2), r → √(x^2 + y^2)} // MatrixForm
```

```
Out[33]= x^i
```

```
Out[34]= ∂x^i/∂x^j
```

```
Out[35]//MatrixForm=
( ∂x^1/∂x^1   ∂x^1/∂x^2 )
( ∂x^2/∂x^1   ∂x^2/∂x^2 )
```

```
Out[36]//MatrixForm=
( Cos[ϕ]   -r Sin[ϕ] )
( Sin[ϕ]   r Cos[ϕ]  )
```

```
Out[37]//MatrixForm=
( x/√(x^2+y^2)   -y )
( y/√(x^2+y^2)   x  )
```

Restore settings.

```
In[38]:= ClearTensorShortcuts[{{x, T}, 1}, {{S, T, δ}, 2}]
```

```
In[39]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors, labs]
```

PartialKroneckerExpand

- `PartialKroneckerExpand[`*δ, order, suborder, expandup : True*`][`*expr*`]` will expand tensors with label *δ*, assumed to be generalized Kroneckers, and with order up and down indices, in terms of *δ*s with suborder and (order-suborder) indices.

The expansion is done on the first suborder up indices, unless the optional 4th argument is set to False, in which case the first suborder down indices are used.

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`, `KroneckerContract`, `KroneckerAbsorb`, `KroneckerEvaluate`.

Examples

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

To avoid defining all 2^8 combinations of up/down indices...

```
In[2]:= δuuuudddd[i_, j_, k_, l_, r_, s_, t_, u_] :=
         Tensor[δ, {i, j, k, l, Void, Void, Void, Void}, {Void, Void, Void, Void, r, s, t, u}]
        κuuuudddd[i_, j_, k_, l_, r_, s_, t_, u_] :=
         Tensor[κ, {i, j, k, l, Void, Void, Void, Void}, {Void, Void, Void, Void, r, s, t, u}]
```

Here we do expansions of different suborders and, as a check, test them against the full expansion of the initial *δ*. The second line gives the partial expansion. The third line checks if the full expansion of the partial expansion is equal to the full expansion of the initial generalized Kronecker. Here the partial expansion is in terms of first order Kroneckers.

```
In[4]:= gk1 = δuuuudddd[i, j, k, l, r, s, t, u]
        lhs = gk1 // FullKroneckerExpand[δ];
        gk1 // PartialKroneckerExpand[δ, 4, 1]
        rhs = % // FullKroneckerExpand[δ];
        lhs == rhs // ExpandAll
```

```
Out[4]= δ^{ijkl}_{rstu}
```

```
Out[6]= -δ^i_u δ^{jkl}_{rst} + δ^i_t δ^{jkl}_{rsu} - δ^i_s δ^{jkl}_{rtu} + δ^i_r δ^{jkl}_{stu}
```

```
Out[8]= True
```

Here we expand in terms of second order Kroneckers.

```
In[9]:= gk1 = δuuuudddd[i, j, k, l, r, s, t, u]
        lhs = gk1 // FullKroneckerExpand[δ];
        gk1 // PartialKroneckerExpand[δ, 4, 2]
        rhs = % // FullKroneckerExpand[δ];
        lhs == rhs // ExpandAll
```

$$\text{Out[9]=}\quad \delta^{ijkl}{}_{rstu}$$

$$\text{Out[11]=}\quad \delta^{ij}{}_{tu}\delta^{kl}{}_{rs} - \delta^{ij}{}_{su}\delta^{kl}{}_{rt} + \delta^{ij}{}_{st}\delta^{kl}{}_{ru} + \delta^{ij}{}_{ru}\delta^{kl}{}_{st} - \delta^{ij}{}_{rt}\delta^{kl}{}_{su} + \delta^{ij}{}_{rs}\delta^{kl}{}_{tu}$$

```
Out[13]= True
```

Here we expand in terms of 3rd order Kroneckers.

```
In[14]:= gk1 = δuuuudddd[i, j, k, l, r, s, t, u]
         lhs = gk1 // FullKroneckerExpand[δ];
         gk1 // PartialKroneckerExpand[δ, 4, 3]
         rhs = % // FullKroneckerExpand[δ];
         lhs == rhs // ExpandAll
```

$$\text{Out[14]=}\quad \delta^{ijkl}{}_{rstu}$$

$$\text{Out[16]=}\quad \delta^{l}{}_{u}\delta^{ijk}{}_{rst} - \delta^{l}{}_{t}\delta^{ijk}{}_{rsu} + \delta^{l}{}_{s}\delta^{ijk}{}_{rtu} - \delta^{l}{}_{r}\delta^{ijk}{}_{stu}$$

```
Out[18]= True
```

The following first does a 3rd order expansion of the 4th order Kronecker. It then does a 2nd order expansion of the resulting 3rd order Kroneckers. It then does a 1st order expansion of the resulting 2nd order Kroneckers. We then check that this is equal to full expansion of the initial Kronecker.

```
In[19]:= gk1 = δuuuudddd[i, j, k, l, r, s, t, u]
         lhs = gk1 // FullKroneckerExpand[δ];
         gk1 // PartialKroneckerExpand[δ, 4, 3]
         % // PartialKroneckerExpand[δ, 3, 2]
         rhs = % // PartialKroneckerExpand[δ, 2, 1]
         lhs == rhs // ExpandAll
```

$$\text{Out[19]=}\quad \delta^{ijkl}{}_{rstu}$$

$$\text{Out[21]=}\quad \delta^{l}{}_{u}\delta^{ijk}{}_{rst} - \delta^{l}{}_{t}\delta^{ijk}{}_{rsu} + \delta^{l}{}_{s}\delta^{ijk}{}_{rtu} - \delta^{l}{}_{r}\delta^{ijk}{}_{stu}$$

$$\begin{aligned}\text{Out[22]=}\quad & \delta^{l}{}_{u}\left(\delta^{k}{}_{t}\delta^{ij}{}_{rs} - \delta^{k}{}_{s}\delta^{ij}{}_{rt} + \delta^{k}{}_{r}\delta^{ij}{}_{st}\right) - \delta^{l}{}_{t}\left(\delta^{k}{}_{u}\delta^{ij}{}_{rs} - \delta^{k}{}_{s}\delta^{ij}{}_{ru} + \delta^{k}{}_{r}\delta^{ij}{}_{su}\right) + \\ & \delta^{l}{}_{s}\left(\delta^{k}{}_{u}\delta^{ij}{}_{rt} - \delta^{k}{}_{t}\delta^{ij}{}_{ru} + \delta^{k}{}_{r}\delta^{ij}{}_{tu}\right) - \delta^{l}{}_{r}\left(\delta^{k}{}_{u}\delta^{ij}{}_{st} - \delta^{k}{}_{t}\delta^{ij}{}_{su} + \delta^{k}{}_{s}\delta^{ij}{}_{tu}\right)\end{aligned}$$

$$\begin{aligned}\text{Out[23]=}\quad & -\left(\left(-\delta^{i}{}_{u}\delta^{j}{}_{t} + \delta^{i}{}_{t}\delta^{j}{}_{u}\right)\delta^{k}{}_{s} - \left(-\delta^{i}{}_{u}\delta^{j}{}_{s} + \delta^{i}{}_{s}\delta^{j}{}_{u}\right)\delta^{k}{}_{t} + \left(-\delta^{i}{}_{t}\delta^{j}{}_{s} + \delta^{i}{}_{s}\delta^{j}{}_{t}\right)\delta^{k}{}_{u}\right)\delta^{l}{}_{r} + \\ & \left(\left(-\delta^{i}{}_{u}\delta^{j}{}_{t} + \delta^{i}{}_{t}\delta^{j}{}_{u}\right)\delta^{k}{}_{r} - \left(-\delta^{i}{}_{u}\delta^{j}{}_{r} + \delta^{i}{}_{r}\delta^{j}{}_{u}\right)\delta^{k}{}_{t} + \left(-\delta^{i}{}_{t}\delta^{j}{}_{r} + \delta^{i}{}_{r}\delta^{j}{}_{t}\right)\delta^{k}{}_{u}\right)\delta^{l}{}_{s} - \\ & \left(\left(-\delta^{i}{}_{u}\delta^{j}{}_{s} + \delta^{i}{}_{s}\delta^{j}{}_{u}\right)\delta^{k}{}_{r} - \left(-\delta^{i}{}_{u}\delta^{j}{}_{r} + \delta^{i}{}_{r}\delta^{j}{}_{u}\right)\delta^{k}{}_{s} + \left(-\delta^{i}{}_{s}\delta^{j}{}_{r} + \delta^{i}{}_{r}\delta^{j}{}_{s}\right)\delta^{k}{}_{u}\right)\delta^{l}{}_{t} + \\ & \left(\left(-\delta^{i}{}_{t}\delta^{j}{}_{s} + \delta^{i}{}_{s}\delta^{j}{}_{t}\right)\delta^{k}{}_{r} - \left(-\delta^{i}{}_{t}\delta^{j}{}_{r} + \delta^{i}{}_{r}\delta^{j}{}_{t}\right)\delta^{k}{}_{s} + \left(-\delta^{i}{}_{s}\delta^{j}{}_{r} + \delta^{i}{}_{r}\delta^{j}{}_{s}\right)\delta^{k}{}_{t}\right)\delta^{l}{}_{u}\end{aligned}$$

```
Out[24]= True
```

The following does an expansion on the down indices.

```
In[25]:= gk1 = δuuuudddd[i, j, k, l, r, s, t, u]
         lhs = gk1 // FullKroneckerExpand[δ];
         gk1 // PartialKroneckerExpand[δ, 4, 1, False]
         rhs = % // FullKroneckerExpand[δ];
         lhs == rhs // ExpandAll
```

$$\text{Out[25]=}\quad \delta^{ijkl}{}_{rstu}$$

$$\text{Out[27]=}\quad -\delta^{l}{}_{r}\,\delta^{ijk}{}_{stu}+\delta^{k}{}_{r}\,\delta^{ijl}{}_{stu}-\delta^{j}{}_{r}\,\delta^{ikl}{}_{stu}+\delta^{i}{}_{r}\,\delta^{jkl}{}_{stu}$$

```
Out[29]= True
```

The following is a multistep expansion on the down indices.

```
In[30]:= gk1 = δuuuudddd[i, j, k, l, r, s, t, u]
         lhs = gk1 // FullKroneckerExpand[δ];
         gk1 // PartialKroneckerExpand[δ, 4, 3, False]
         % // PartialKroneckerExpand[δ, 3, 2, False]
         rhs = % // PartialKroneckerExpand[δ, 2, 1, False]
         lhs == rhs // ExpandAll
```

$$\text{Out[30]=}\quad \delta^{ijkl}{}_{rstu}$$

$$\text{Out[32]=}\quad \delta^{l}{}_{u}\,\delta^{ijk}{}_{rst}-\delta^{k}{}_{u}\,\delta^{ijl}{}_{rst}+\delta^{j}{}_{u}\,\delta^{ikl}{}_{rst}-\delta^{i}{}_{u}\,\delta^{jkl}{}_{rst}$$

$$\text{Out[33]=}\quad \delta^{l}{}_{u}\left(\delta^{k}{}_{t}\,\delta^{ij}{}_{rs}-\delta^{j}{}_{t}\,\delta^{ik}{}_{rs}+\delta^{i}{}_{t}\,\delta^{jk}{}_{rs}\right)-\delta^{k}{}_{u}\left(\delta^{l}{}_{t}\,\delta^{ij}{}_{rs}-\delta^{j}{}_{t}\,\delta^{il}{}_{rs}+\delta^{i}{}_{t}\,\delta^{jl}{}_{rs}\right)+\delta^{j}{}_{u}\left(\delta^{l}{}_{t}\,\delta^{ik}{}_{rs}-\delta^{k}{}_{t}\,\delta^{il}{}_{rs}+\delta^{i}{}_{t}\,\delta^{kl}{}_{rs}\right)-\delta^{i}{}_{u}\left(\delta^{l}{}_{t}\,\delta^{jk}{}_{rs}-\delta^{k}{}_{t}\,\delta^{jl}{}_{rs}+\delta^{j}{}_{t}\,\delta^{kl}{}_{rs}\right)$$

$$\begin{aligned}\text{Out[34]=}\quad & -\delta^{k}{}_{u}\left(-\delta^{j}{}_{t}\left(-\delta^{i}{}_{s}\,\delta^{l}{}_{r}+\delta^{i}{}_{r}\,\delta^{l}{}_{s}\right)+\delta^{i}{}_{t}\left(-\delta^{j}{}_{s}\,\delta^{l}{}_{r}+\delta^{j}{}_{r}\,\delta^{l}{}_{s}\right)+\left(-\delta^{i}{}_{s}\,\delta^{j}{}_{r}+\delta^{i}{}_{r}\,\delta^{j}{}_{s}\right)\delta^{l}{}_{t}\right)+\\ & \delta^{j}{}_{u}\left(-\delta^{k}{}_{t}\left(-\delta^{i}{}_{s}\,\delta^{l}{}_{r}+\delta^{i}{}_{r}\,\delta^{l}{}_{s}\right)+\delta^{i}{}_{t}\left(-\delta^{k}{}_{s}\,\delta^{l}{}_{r}+\delta^{k}{}_{r}\,\delta^{l}{}_{s}\right)+\left(-\delta^{i}{}_{s}\,\delta^{k}{}_{r}+\delta^{i}{}_{r}\,\delta^{k}{}_{s}\right)\delta^{l}{}_{t}\right)-\\ & \delta^{i}{}_{u}\left(-\delta^{k}{}_{t}\left(-\delta^{j}{}_{s}\,\delta^{l}{}_{r}+\delta^{j}{}_{r}\,\delta^{l}{}_{s}\right)+\delta^{j}{}_{t}\left(-\delta^{k}{}_{s}\,\delta^{l}{}_{r}+\delta^{k}{}_{r}\,\delta^{l}{}_{s}\right)+\left(-\delta^{j}{}_{s}\,\delta^{k}{}_{r}+\delta^{j}{}_{r}\,\delta^{k}{}_{s}\right)\delta^{l}{}_{t}\right)+\\ & \left(-\delta^{j}{}_{t}\left(-\delta^{i}{}_{s}\,\delta^{k}{}_{r}+\delta^{i}{}_{r}\,\delta^{k}{}_{s}\right)+\delta^{i}{}_{t}\left(-\delta^{j}{}_{s}\,\delta^{k}{}_{r}+\delta^{j}{}_{r}\,\delta^{k}{}_{s}\right)+\left(-\delta^{i}{}_{s}\,\delta^{j}{}_{r}+\delta^{i}{}_{r}\,\delta^{j}{}_{s}\right)\delta^{k}{}_{t}\right)\delta^{l}{}_{u}\end{aligned}$$

```
Out[35]= True
```

In the following we alternately expand on the down and up indices.

```
In[36]:= gk1 = δuuuudddd[i, j, k, l, r, s, t, u]
         lhs = gk1 // FullKroneckerExpand[δ];
         gk1 // PartialKroneckerExpand[δ, 4, 3, False]
         % // PartialKroneckerExpand[δ, 3, 2, True]
         rhs = % // PartialKroneckerExpand[δ, 2, 1, False]
         lhs == rhs // ExpandAll
```

$$\text{Out[36]=}\ \delta^{ijkl}{}_{rstu}$$

$$\text{Out[38]=}\ \delta^{l}{}_{u}\,\delta^{ijk}{}_{rst}-\delta^{k}{}_{u}\,\delta^{ijl}{}_{rst}+\delta^{j}{}_{u}\,\delta^{ikl}{}_{rst}-\delta^{i}{}_{u}\,\delta^{jkl}{}_{rst}$$

$$\text{Out[39]=}\ \delta^{l}{}_{u}\left(\delta^{k}{}_{t}\,\delta^{ij}{}_{rs}-\delta^{k}{}_{s}\,\delta^{ij}{}_{rt}+\delta^{k}{}_{r}\,\delta^{ij}{}_{st}\right)-\delta^{k}{}_{u}\left(\delta^{l}{}_{t}\,\delta^{ij}{}_{rs}-\delta^{l}{}_{s}\,\delta^{ij}{}_{rt}+\delta^{l}{}_{r}\,\delta^{ij}{}_{st}\right)+\delta^{j}{}_{u}\left(\delta^{l}{}_{t}\,\delta^{ik}{}_{rs}-\delta^{l}{}_{s}\,\delta^{ik}{}_{rt}+\delta^{l}{}_{r}\,\delta^{ik}{}_{st}\right)-\delta^{i}{}_{u}\left(\delta^{l}{}_{t}\,\delta^{jk}{}_{rs}-\delta^{l}{}_{s}\,\delta^{jk}{}_{rt}+\delta^{l}{}_{r}\,\delta^{jk}{}_{st}\right)$$

$$\begin{aligned}\text{Out[40]=}\ &-\delta^{k}{}_{u}\left(\left(-\delta^{i}{}_{t}\,\delta^{j}{}_{s}+\delta^{i}{}_{s}\,\delta^{j}{}_{t}\right)\delta^{l}{}_{r}-\left(-\delta^{i}{}_{t}\,\delta^{j}{}_{r}+\delta^{i}{}_{r}\,\delta^{j}{}_{t}\right)\delta^{l}{}_{s}+\left(-\delta^{i}{}_{s}\,\delta^{j}{}_{r}+\delta^{i}{}_{r}\,\delta^{j}{}_{s}\right)\delta^{l}{}_{t}\right)+\\ &\delta^{j}{}_{u}\left(\left(-\delta^{i}{}_{t}\,\delta^{k}{}_{s}+\delta^{i}{}_{s}\,\delta^{k}{}_{t}\right)\delta^{l}{}_{r}-\left(-\delta^{i}{}_{t}\,\delta^{k}{}_{r}+\delta^{i}{}_{r}\,\delta^{k}{}_{t}\right)\delta^{l}{}_{s}+\left(-\delta^{i}{}_{s}\,\delta^{k}{}_{r}+\delta^{i}{}_{r}\,\delta^{k}{}_{s}\right)\delta^{l}{}_{t}\right)-\\ &\delta^{i}{}_{u}\left(\left(-\delta^{j}{}_{t}\,\delta^{k}{}_{s}+\delta^{j}{}_{s}\,\delta^{k}{}_{t}\right)\delta^{l}{}_{r}-\left(-\delta^{j}{}_{t}\,\delta^{k}{}_{r}+\delta^{j}{}_{r}\,\delta^{k}{}_{t}\right)\delta^{l}{}_{s}+\left(-\delta^{j}{}_{s}\,\delta^{k}{}_{r}+\delta^{j}{}_{r}\,\delta^{k}{}_{s}\right)\delta^{l}{}_{t}\right)+\\ &\left(\left(-\delta^{i}{}_{t}\,\delta^{j}{}_{s}+\delta^{i}{}_{s}\,\delta^{j}{}_{t}\right)\delta^{k}{}_{r}-\left(-\delta^{i}{}_{t}\,\delta^{j}{}_{r}+\delta^{i}{}_{r}\,\delta^{j}{}_{t}\right)\delta^{k}{}_{s}+\left(-\delta^{i}{}_{s}\,\delta^{j}{}_{r}+\delta^{i}{}_{r}\,\delta^{j}{}_{s}\right)\delta^{k}{}_{t}\right)\delta^{l}{}_{u}\end{aligned}$$

```
Out[41]= True
```

Labels other than δ can be used to represent the generalized Kronecker.

```
In[42]:= gk1 = κuuuudddd[i, j, k, l, r, s, t, u]
         lhs = gk1 // FullKroneckerExpand[κ];
         gk1 // PartialKroneckerExpand[κ, 4, 3, False]
         % // PartialKroneckerExpand[κ, 3, 2, True]
         rhs = % // PartialKroneckerExpand[κ, 2, 1, False]
         lhs == rhs // ExpandAll
```

$$\text{Out[42]=}\ \kappa^{ijkl}{}_{rstu}$$

$$\text{Out[44]=}\ \kappa^{l}{}_{u}\,\kappa^{ijk}{}_{rst}-\kappa^{k}{}_{u}\,\kappa^{ijl}{}_{rst}+\kappa^{j}{}_{u}\,\kappa^{ikl}{}_{rst}-\kappa^{i}{}_{u}\,\kappa^{jkl}{}_{rst}$$

$$\text{Out[45]=}\ \kappa^{l}{}_{u}\left(\kappa^{k}{}_{t}\,\kappa^{ij}{}_{rs}-\kappa^{k}{}_{s}\,\kappa^{ij}{}_{rt}+\kappa^{k}{}_{r}\,\kappa^{ij}{}_{st}\right)-\kappa^{k}{}_{u}\left(\kappa^{l}{}_{t}\,\kappa^{ij}{}_{rs}-\kappa^{l}{}_{s}\,\kappa^{ij}{}_{rt}+\kappa^{l}{}_{r}\,\kappa^{ij}{}_{st}\right)+\kappa^{j}{}_{u}\left(\kappa^{l}{}_{t}\,\kappa^{ik}{}_{rs}-\kappa^{l}{}_{s}\,\kappa^{ik}{}_{rt}+\kappa^{l}{}_{r}\,\kappa^{ik}{}_{st}\right)-\kappa^{i}{}_{u}\left(\kappa^{l}{}_{t}\,\kappa^{jk}{}_{rs}-\kappa^{l}{}_{s}\,\kappa^{jk}{}_{rt}+\kappa^{l}{}_{r}\,\kappa^{jk}{}_{st}\right)$$

$$\begin{aligned}\text{Out[46]=}\ &-\kappa^{k}{}_{u}\left(\left(-\kappa^{i}{}_{t}\,\kappa^{j}{}_{s}+\kappa^{i}{}_{s}\,\kappa^{j}{}_{t}\right)\kappa^{l}{}_{r}-\left(-\kappa^{i}{}_{t}\,\kappa^{j}{}_{r}+\kappa^{i}{}_{r}\,\kappa^{j}{}_{t}\right)\kappa^{l}{}_{s}+\left(-\kappa^{i}{}_{s}\,\kappa^{j}{}_{r}+\kappa^{i}{}_{r}\,\kappa^{j}{}_{s}\right)\kappa^{l}{}_{t}\right)+\\ &\kappa^{j}{}_{u}\left(\left(-\kappa^{i}{}_{t}\,\kappa^{k}{}_{s}+\kappa^{i}{}_{s}\,\kappa^{k}{}_{t}\right)\kappa^{l}{}_{r}-\left(-\kappa^{i}{}_{t}\,\kappa^{k}{}_{r}+\kappa^{i}{}_{r}\,\kappa^{k}{}_{t}\right)\kappa^{l}{}_{s}+\left(-\kappa^{i}{}_{s}\,\kappa^{k}{}_{r}+\kappa^{i}{}_{r}\,\kappa^{k}{}_{s}\right)\kappa^{l}{}_{t}\right)-\\ &\kappa^{i}{}_{u}\left(\left(-\kappa^{j}{}_{t}\,\kappa^{k}{}_{s}+\kappa^{j}{}_{s}\,\kappa^{k}{}_{t}\right)\kappa^{l}{}_{r}-\left(-\kappa^{j}{}_{t}\,\kappa^{k}{}_{r}+\kappa^{j}{}_{r}\,\kappa^{k}{}_{t}\right)\kappa^{l}{}_{s}+\left(-\kappa^{j}{}_{s}\,\kappa^{k}{}_{r}+\kappa^{j}{}_{r}\,\kappa^{k}{}_{s}\right)\kappa^{l}{}_{t}\right)+\\ &\left(\left(-\kappa^{i}{}_{t}\,\kappa^{j}{}_{s}+\kappa^{i}{}_{s}\,\kappa^{j}{}_{t}\right)\kappa^{k}{}_{r}-\left(-\kappa^{i}{}_{t}\,\kappa^{j}{}_{r}+\kappa^{i}{}_{r}\,\kappa^{j}{}_{t}\right)\kappa^{k}{}_{s}+\left(-\kappa^{i}{}_{s}\,\kappa^{j}{}_{r}+\kappa^{i}{}_{r}\,\kappa^{j}{}_{s}\right)\kappa^{k}{}_{t}\right)\kappa^{l}{}_{u}\end{aligned}$$

```
Out[47]= True
```

```
In[48]:= δuuuudddd[i_, j_, k_, l_, r_, s_, t_, u_] =.
         κuuuudddd[i_, j_, k_, l_, r_, s_, t_, u_] =.
```

PartialSum

- `PartialSum[`*baseindex, newindices*`][`*expr*`]` will expand the dummy indices of a tensor expression into components that have one or more baseindex indices and an unexpanded remainder. Symbolic indices are replaced with newindices.

The newindices list of symbols is used to replace the existing dummy index symbols. Greek indices might be replaced with Latin indices to indicate a restricted further expansion. But it is up to the user to actually used a restriced expansion when further expansions are perform.

It is also possible to use different flavor indices for the unexpanded portion, and then assign a restricted set of base indices to the flavor.

Any flavor must be on the newindices and the baseindex.

PartialSum is mapped over arrays, equations and sums.

See also: `PartialArray`, `EinsteinSum`, `EinsteinArray`, `SumExpansion`.

Examples

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

Save the settings and declare base indices and flavors.

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareBaseIndices[{0, 1, 2, 3}]
        DeclareIndexFlavor[{red, Red}, {blue, Blue}];

In[7]:= DefineTensorShortcuts[{{dx, λ}, 1}, {{g, T}, 2}]
```

The following expands a contracted tensor on the 0 index.

```
In[8]:= Tud[μ, μ]
        % // PartialSum[0, {i}]

Out[8]= T^μ_μ

Out[9]= T^0_0 + T^i_i
```

The flavor must be on both the base index and the new index.

```
In[10]:= Tud[μ, μ] // ToFlavor[red]
         % // PartialSum[red@0, {red@i}]

Out[10]= T^μ_μ

Out[11]= T^0_0 + T^i_i
```

The following first does an array expansion of a tensor expression on the 0 index, then does a sum expansion on the 0 index and then does a complete expansion.

```
In[12]:= Tuu[μ, ν] λd[ν] // ToFlavor[red]
         % // PartialArray[red@0, {red@i}]
         % // PartialSum[red@0, {red@j}]
         EinsteinArray[{1, 2, 3}] /@ EinsteinSum[{1, 2, 3}] /@ %
```

$$\text{Out[12]=}\quad T^{\mu\,\nu}\,\lambda_{\nu}$$

$$\text{Out[13]=}\quad \{T^{0\,\nu}\,\lambda_{\nu},\ T^{i\,\nu}\,\lambda_{\nu}\}$$

$$\text{Out[14]=}\quad \left\{T^{0\,0}\,\lambda_{0}+T^{0\,j}\,\lambda_{j},\ T^{i\,0}\,\lambda_{0}+T^{i\,j}\,\lambda_{j}\right\}$$

$$\text{Out[15]=}\quad \{T^{0\,0}\,\lambda_{0}+T^{0\,1}\,\lambda_{1}+T^{0\,2}\,\lambda_{2}+T^{0\,3}\,\lambda_{3},\ \{T^{1\,0}\,\lambda_{0}+T^{1\,1}\,\lambda_{1}+T^{1\,2}\,\lambda_{2}+T^{1\,3}\,\lambda_{3},\ T^{2\,0}\,\lambda_{0}+T^{2\,1}\,\lambda_{1}+T^{2\,2}\,\lambda_{2}+T^{2\,3}\,\lambda_{3},\ T^{3\,0}\,\lambda_{0}+T^{3\,1}\,\lambda_{1}+T^{3\,2}\,\lambda_{2}+T^{3\,3}\,\lambda_{3}\}\}$$

The following expands the metric line element into temporal and spatial parts.

```
In[16]:= gdd[μ, ν] dxu[μ] dxu[ν]
         % // PartialSum[0, {i, j}]
```

$$\text{Out[16]=}\quad dx^{\mu}\,dx^{\nu}\,g_{\mu\,\nu}$$

$$\text{Out[17]=}\quad (dx^{0})^{2}\,g_{0\,0}+dx^{0}\,dx^{j}\,g_{0\,j}+dx^{0}\,dx^{i}\,g_{i\,0}+dx^{i}\,dx^{j}\,g_{i\,j}$$

The new indices must be sufficient to replace the old dummy indices after removing any conflicts with free indices.

```
In[18]:= gdd[μ, ν] Tuu[i, μ] λu[ν] // ToFlavor[red]
         % // PartialSum[red@0, red /@ {i, j}]
```

$$\text{Out[18]=}\quad g_{\mu\,\nu}\,T^{i\,\mu}\,\lambda^{\nu}$$

```
PartialSum::newindices : New indices {j}, after
   removing freeindices {i}, are insufficient to replace {μ, ν}
```

$$\text{Out[19]=}\quad \$\text{Aborted}$$

The routine works with multiple base index sets.

```
In[20]:= DeclareBaseIndices[{t, x, y, z}, {red, {t, ρ, θ, ϕ}}, {blue, {ρ, θ, ϕ}}]
```

```
In[21]:= Tud[μ, μ]
         % // PartialSum[t, {i}]
         MapAt[EinsteinSum[{x, y, z}], %, 1]
```

$$\text{Out[21]=}\quad T^{\mu}{}_{\mu}$$

$$\text{Out[22]=}\quad T^{i}{}_{i}+T^{t}{}_{t}$$

$$\text{Out[23]=}\quad T^{t}{}_{t}+T^{x}{}_{x}+T^{y}{}_{y}+T^{z}{}_{z}$$

```
In[24]:= Tud[μ, μ] // ToFlavor[red]
         % // PartialSum[red@t, {red@i}]
         MapAt[EinsteinSum[{ρ, θ, ϕ}], %, 1]
```

$$\text{Out[24]=}\quad T^{\mu}{}_{\mu}$$

$$\text{Out[25]=}\quad T^{i}{}_{i}+T^{t}{}_{t}$$

$$\text{Out[26]=}\quad T^{t}{}_{t}+T^{\theta}{}_{\theta}+T^{\rho}{}_{\rho}+T^{\phi}{}_{\phi}$$

In the `DeclareBaseIndices` statement above, we have associated the blue flavor with the restricted set of spatial indices. By replacing the red μ with a blue μ we can do a simple `EinsteinSum` to complete the expansion.

```
In[27]:= Tud[μ, μ] // ToFlavor[red]
         % // PartialSum[red@t, {blue@μ}]
         % // EinsteinSum[]
```

$$\text{Out[27]=}\quad T^{\mu}{}_{\mu}$$

$$\text{Out[28]=}\quad T^{\mu}{}_{\mu} + T^{t}{}_{t}$$

$$\text{Out[29]=}\quad T^{\theta}{}_{\theta} + T^{\rho}{}_{\rho} + T^{\phi}{}_{\phi} + T^{t}{}_{t}$$

Only indices of the baseindex flavor are broken out.

```
In[30]:= Tuu[red@μ, ν] λd[red@μ] dxd[ν]
         % // PartialSum[red@t, {red@i}]
```

$$\text{Out[30]=}\quad dx_{\nu}\, T^{\mu\,\nu}\, \lambda_{\mu}$$

$$\text{Out[31]=}\quad dx_{\nu}\, T^{i\,\nu}\, \lambda_{i} + dx_{\nu}\, T^{t\,\nu}\, \lambda_{t}$$

```
In[32]:= Tuu[red@μ, ν] λd[red@μ] dxd[ν]
         % // PartialSum[t, {i}]
```

$$\text{Out[32]=}\quad dx_{\nu}\, T^{\mu\,\nu}\, \lambda_{\mu}$$

$$\text{Out[33]=}\quad dx_{i}\, T^{\mu\,i}\, \lambda_{\mu} + dx_{t}\, T^{\mu\,t}\, \lambda_{\mu}$$

Restore the initial values...

```
In[34]:= ClearTensorShortcuts[{{dx, λ}, 1}, {{g, T}, 2}]
```

```
In[35]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

PatternReplaceIndex

- `PatternReplaceIndex[`*newindexlist, pattern, checkdummies : True*`][`*expr*`]` will replace dummy indices in simple terms if the terms match the pattern with named pattern indices.

The object is to change all matching expressions to ones with the same dummy indices.

If the head of an expression is Plus, the routine is mapped onto the terms.

A simple term is a Tensor or a weighted product of tensors.

The index flavors of a matching term must match the flavors of the new indices.

With the default value of checkdummies, the routine aborts if the pattern indices are not dummy indices in the matching expression. This can be overridden by setting the third optional argument to False. This can produce incorrect results if not done with care but can be useful in writing simpler patterns.

IndexChange can be used to perform specific reindexing under the user's control. SimplifyTensorSum, sometimes used with MapLevelParts is another way to simplify expressions.

The simplification routines are not high powered. They do not take into account symmetries. See `SymmetrizeSlots` to apply some symmetries.

See also: `UpDownSwap`, `IndexChange`, `SymmetrizeSlots`, `NondependentPartialD`, `SimplifyTensorSum`, `MapLevelParts`.

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

The following expression is simplified by converting all terms to the same set of dummy indices.

```
In[6]:= a Fdduu[i, j, i, j] + b Fdduu[j, p, j, p] + c Fdduu[m, r, m, r]
        % // PatternReplaceIndex[{μ, ν}, Fdduu[m_, n_, m_, n_]]
        % // Factor

Out[6]= a F_ij^ij + b F_jp^jp + c F_mr^mr

Out[7]= a F_μν^μν + b F_μν^μν + c F_μν^μν

Out[8]= (a + b + c) F_μν^μν
```

`SimplifyTensorSum` will also perform the simplification but with no direct control over the choice of dummy index.

```
In[9]:= a Fdduu[i, j, i, j] + b Fdduu[j, p, j, p] + c Fdduu[m, r, m, r]
        % // SimplifyTensorSum // Factor
```

Out[9]= $a F_{ij}{}^{ij} + b F_{jp}{}^{jp} + c F_{mr}{}^{mr}$

Out[10]= $(a + b + c) F_{ij}{}^{ij}$

SimplifyTensorSum doesn't simplify the following expression because it replaces dummy indices in order of the factors and gets different results for the two terms.

```
In[11]:= a Buu[i, j] Gd[i] + b Buu[k, j] Gd[k] Aud[α, α]
         % // SimplifyTensorSum // Factor
```

Out[11]= $a B^{ij} G_i + b A^{\alpha}{}_{\alpha} B^{kj} G_k$

Out[12]= $a B^{ij} G_i + b A^{i}{}_{i} B^{\alpha j} G_{\alpha}$

With PatternReplaceIndex we can pick out the relevant factors in the two terms and the simplification is performed.

```
In[13]:= a Buu[i, j] Gd[i] + b Buu[k, j] Gd[k] Aud[α, α]
         % // PatternReplaceIndex[{μ}, Buu[i_, j] Gd[i_]] // Factor
```

Out[13]= $a B^{ij} G_i + b A^{\alpha}{}_{\alpha} B^{kj} G_k$

Out[14]= $(a + b A^{\alpha}{}_{\alpha}) B^{\mu j} G_{\mu}$

The routine does not replace tensor labels or symbols elsewhere.

```
In[15]:= a Auu[A, j] Gd[A] + A Auu[k, j] Gd[k]
         % // PatternReplaceIndex[{μ}, Auu[i_, j] Gd[i_]] // Factor
```

Out[15]= $a A^{Aj} G_A + A A^{kj} G_k$

Out[16]= $(a + A) A^{\mu j} G_{\mu}$

The following does not work because the routine cannot check if the indices in the pattern are dummies.

```
In[17]:= (addd[r, s, t] + addd[s, r, t] + addd[s, t, r]) bu[r] bu[s] bu[t] // ToFlavor[red]
         Expand[%] // PatternReplaceIndex[{μ, ν, σ}, addd[r_, s_, t_]]
```

Out[17]= $(a_{rst} + a_{srt} + a_{str}) b^r b^s b^t$

```
PatternReplaceIndex::dummies : Some pattern indices in a_r_s_t_ are not dummies.
```

```
Out[18]= $Aborted
```

But by setting the optional argument, checkdummies, to False we can suppress the dummy checking.

```
In[19]:= (addd[r, s, t] + addd[s, r, t] + addd[s, t, r]) bu[r] bu[s] bu[t] // ToFlavor[red]
         % // PatternReplaceIndex[red /@ {μ, ν, σ}, addd[a_, b_, c_], False]
```

Out[19]= $(a_{rst} + a_{srt} + a_{str}) b^r b^s b^t$

Out[20]= $3 a_{\mu\nu\sigma} b^{\mu} b^{\nu} b^{\sigma}$

The order of index replacement is affected by the sort order of the named indices in the pattern. The named indices are sorted and then matched with the new index list.

```
In[21]:= (addd[r, s, t] + addd[s, r, t] + addd[s, t, r]) bu[r] bu[s] bu[t] // ToFlavor[red]
         % // PatternReplaceIndex[red /@ {μ, ν, σ}, addd[c_, b_, a_], False]
```

$$\text{Out[21]=}\quad (a_{r\,s\,t} + a_{s\,r\,t} + a_{s\,t\,r})\; b^{r}\; b^{s}\; b^{t}$$

$$\text{Out[22]=}\quad 3\; a_{\sigma\,\nu\,\mu}\; b^{\mu}\; b^{\nu}\; b^{\sigma}$$

If the flavors are not the same there is no match.

```
In[23]:= (addd[r, s, t] + addd[s, r, t] + addd[s, t, r]) (bu[r] bu[s] bu[t])
         % // PatternReplaceIndex[red /@ {μ, ν, σ}, addd[t_, s_, r_], False]
```

$$\text{Out[23]=}\quad (a_{r\,s\,t} + a_{s\,r\,t} + a_{s\,t\,r})\; b^{r}\; b^{s}\; b^{t}$$

$$\text{Out[24]=}\quad (a_{r\,s\,t} + a_{s\,r\,t} + a_{s\,t\,r})\; b^{r}\; b^{s}\; b^{t}$$

Non-pattern indices are only replaced if they have the correct flavor. So the following would be incorrect and illustrates the danger of not checking dummy indices.

```
In[25]:= (addd[r, s, t] + addd[s, r, t] + addd[s, t, r] // ToFlavor[red]) (bu[r] bu[s] bu[t])
         % // PatternReplaceIndex[red /@ {μ, ν, σ}, addd[t_, s_, r_], False]
```

$$\text{Out[25]=}\quad (a_{r\,s\,t} + a_{s\,r\,t} + a_{s\,t\,r})\; b^{r}\; b^{s}\; b^{t}$$

$$\text{Out[26]=}\quad 3\; a_{\sigma\,\nu\,\mu}\; b^{r}\; b^{s}\; b^{t}$$

```
In[27]:= a Auu[i, j] CovariantD[Gd[i], ν] + b Auu[k, j] CovariantD[Gd[k], ν]
         % // PatternReplaceIndex[{μ}, Auu[i_, j] CovariantD[Gd[i_], ν]] // Factor
```

$$\text{Out[27]=}\quad a\; G_{i\,;\,\nu}\; A^{i\,j} + b\; G_{k\,;\,\nu}\; A^{k\,j}$$

$$\text{Out[28]=}\quad (a + b)\; G_{\mu\,;\,\nu}\; A^{\mu\,j}$$

The following does not work because the generated pattern is not a simple term.

```
In[29]:= a Auu[i, j] CovariantD[Gd[i], ν] + b Auu[k, j] CovariantD[Gd[k], ν]
         % // ExpandCovariantD[{x, δ, g, Γ}, a]
         % // PatternReplaceIndex[{μ},
           Auu[i_, j] (CovariantD[Gd[i_], ν] // ExpandCovariantD[{x, δ, g, Γ}, a]), False]
```

$$\text{Out[29]=}\quad a\; G_{i\,;\,\nu}\; A^{i\,j} + b\; G_{k\,;\,\nu}\; A^{k\,j}$$

$$\text{Out[30]=}\quad a\; A^{i\,j} \left(-G_{a}\; \Gamma^{a}{}_{\nu\,i} + \frac{\partial G_{i}}{\partial x^{\nu}}\right) + b\; A^{k\,j} \left(-G_{a}\; \Gamma^{a}{}_{\nu\,k} + \frac{\partial G_{k}}{\partial x^{\nu}}\right)$$

```
RawIndex::notindex : i_ is not a Symbol, Integer or Flavor.

ExpandCovariantD::nottensor :
 A covariant derivative , G_{i_;ν}, cannot be expanded because
   Tensorial cannot assess the tensor nature of the expression.
```

```
Out[31]= $Aborted
```

It could be simplified with `SimplifyTensorSum`.

```
In[32]:= a Auu[i, j] CovariantD[Gd[i], ν] + b Auu[k, j] CovariantD[Gd[k], ν]
         % // ExpandCovariantD[{x, δ, g, Γ}, a]
         % // SimplifyTensorSum // Simplify
         MapAt[Minus, %, {{1}, {4}}]
```

$$\text{Out[32]=}\quad a\,G_{i;\nu}\,A^{i\,j} + b\,G_{k;\nu}\,A^{k\,j}$$

$$\text{Out[33]=}\quad a\,A^{i\,j}\left(-G_a\,\Gamma^{a}{}_{\nu\,i} + \frac{\partial G_i}{\partial x^{\nu}}\right) + b\,A^{k\,j}\left(-G_a\,\Gamma^{a}{}_{\nu\,k} + \frac{\partial G_k}{\partial x^{\nu}}\right)$$

$$\text{Out[34]=}\quad -(a+b)\,A^{i\,j}\left(G_a\,\Gamma^{a}{}_{\nu\,i} - \frac{\partial G_i}{\partial x^{\nu}}\right)$$

$$\text{Out[35]=}\quad (a+b)\,A^{i\,j}\left(-G_a\,\Gamma^{a}{}_{\nu\,i} + \frac{\partial G_i}{\partial x^{\nu}}\right)$$

Restore old settings...

```
In[36]:= ClearTensorShortcuts[{{A, G, b}, 1}, {{A, B, S}, 2}, {{a, F}, 3}, {F, 4}]
```

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

PermutationPseudotensor

- `PermutationPseudotensor[n]` will generate the array of values that correspond to the completely antisymmetric n-dimensional tensor.

The Levi-Civita tensor may be constructed using the PermutationPseudotensor

See also: `FullKroneckerExpand`, `PartialKroneckerExpand`, `ContractKronecker`.

Examples

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

Here is the 3-dimensional permutation tensor.

```
In[2]:= PermutationPseudotensor[3] // MatrixForm
```

```
Out[2]//MatrixForm=
```

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

Here is the 4-dimensional permutation tensor.

```
In[3]:= PermutationPseudotensor[4] // MatrixForm
```

```
Out[3]//MatrixForm=
```

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

PermutationSymbolRule

- `PermutationSymbolRule[ϵ]` creates a set of rules that will substitute values for a permutation pseudotensor symbol with label `ϵ`.

For substitution the indices must be all up or all down, must be base indices and must all be in the same flavor.

A symbol with the indices in the declared order of `BaseIndices` will have a value of `1`.

An alternative is to set the values to the PermutationPseudotensor.

See also: `FullKroneckerExpand`, `PartialKroneckerExpand`, `PermutationPseudotensor`.

Examples

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

Save settings.

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

```
In[4]:= DeclareBaseIndices[{1, 2, 3}]
        DeclareIndexFlavor[{red, Red}]
        DefineTensorShortcuts[ϵ, 3]
```

```
In[7]:= ϵddd[2, 1, 3]
        % /. PermutationSymbolRule[ϵ]
```

$$Out[7]= \epsilon_{2\,1\,3}$$

$$Out[8]= -1$$

The following shows results for various cases. Notice the cases that don't substitute.

```
In[9]:= MapThread[{#, # /. PermutationSymbolRule[ϵ]} &,
          {{ϵddd[a, c, b], ϵuuu[a, b, c] // ToFlavor[red],
            ϵddd[2, 3, 1], ϵddd[2, 3, 1] // ToFlavor[red], ϵddd[red@2, 3, 1],
            ϵddd[1, 3, 2], ϵddd[1, 2, 2], ϵdud[1, 3, 2]}}] // TableForm
```

Out[9]//TableForm=

$\epsilon_{a\,c\,b}$	$\epsilon_{a\,c\,b}$
$\epsilon^{a\,b\,c}$	$\epsilon^{a\,b\,c}$
$\epsilon_{2\,3\,1}$	1
$\epsilon_{2\,3\,1}$	1
$\epsilon_{2\,3\,1}$	$\epsilon_{2\,3\,1}$
$\epsilon_{1\,3\,2}$	-1
$\epsilon_{1\,2\,2}$	0
$\epsilon_{1}{}^{3}{}_{2}$	$\epsilon_{1}{}^{3}{}_{2}$

An expansion of the array is the same as the PermutationPseudotensor.

```
In[10]:= ϵddd[i, j, k]
         % // ToArrayValues[] // MatrixForm
         % /. PermutationSymbolRule[ϵ] // MatrixForm
         % == PermutationPseudotensor[3]
```

Out[10]= ϵ_{ijk}

Out[11]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} \epsilon_{111} \\ \epsilon_{112} \\ \epsilon_{113} \end{pmatrix} & \begin{pmatrix} \epsilon_{121} \\ \epsilon_{122} \\ \epsilon_{123} \end{pmatrix} & \begin{pmatrix} \epsilon_{131} \\ \epsilon_{132} \\ \epsilon_{133} \end{pmatrix} \\ \begin{pmatrix} \epsilon_{211} \\ \epsilon_{212} \\ \epsilon_{213} \end{pmatrix} & \begin{pmatrix} \epsilon_{221} \\ \epsilon_{222} \\ \epsilon_{223} \end{pmatrix} & \begin{pmatrix} \epsilon_{231} \\ \epsilon_{232} \\ \epsilon_{233} \end{pmatrix} \\ \begin{pmatrix} \epsilon_{311} \\ \epsilon_{312} \\ \epsilon_{313} \end{pmatrix} & \begin{pmatrix} \epsilon_{321} \\ \epsilon_{322} \\ \epsilon_{323} \end{pmatrix} & \begin{pmatrix} \epsilon_{331} \\ \epsilon_{332} \\ \epsilon_{333} \end{pmatrix} \end{pmatrix}$$

Out[12]//MatrixForm=

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

Out[13]= True

Here is a case of base indices {q, p, r} that are not in the natural sort order.

```
In[14]:= DeclareBaseIndices[{q, p, r}]
         MapThread[{#, # /. PermutationSymbolRule[ϵ]} &,
           {{ϵddd[a, c, b], ϵuuu[a, b, c] // ToFlavor[red],
             ϵddd[q, p, r], ϵddd[p, r, q] // ToFlavor[red], ϵddd[red@p, r, q],
             ϵddd[p, q, r], ϵddd[p, r, r], ϵdud[p, q, r]}}] // TableForm
```

Out[15]//TableForm=

ϵ_{acb}	ϵ_{acb}
ϵ^{abc}	ϵ^{abc}
ϵ_{qpr}	1
ϵ_{prq}	1
ϵ_{prq}	ϵ_{prq}
ϵ_{pqr}	-1
ϵ_{prr}	0
$\epsilon_{p}{}^{q}{}_{r}$	$\epsilon_{p}{}^{q}{}_{r}$

It is still equal to the pseudotensor.

```
In[16]:= ϵddd[i, j, k]
         % // ToArrayValues[] // MatrixForm
         % /. PermutationSymbolRule[ϵ] // MatrixForm
         % == PermutationPseudotensor[3]
         DeclareBaseIndices[{1, 2, 3}]
```

Out[16]= ϵ_{ijk}

Out[17]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} \epsilon_{qqq} \\ \epsilon_{qqp} \\ \epsilon_{qqr} \end{pmatrix} & \begin{pmatrix} \epsilon_{qpq} \\ \epsilon_{qpp} \\ \epsilon_{qpr} \end{pmatrix} & \begin{pmatrix} \epsilon_{qrq} \\ \epsilon_{qrp} \\ \epsilon_{qrr} \end{pmatrix} \\ \begin{pmatrix} \epsilon_{pqq} \\ \epsilon_{pqp} \\ \epsilon_{pqr} \end{pmatrix} & \begin{pmatrix} \epsilon_{ppq} \\ \epsilon_{ppp} \\ \epsilon_{ppr} \end{pmatrix} & \begin{pmatrix} \epsilon_{prq} \\ \epsilon_{prp} \\ \epsilon_{prr} \end{pmatrix} \\ \begin{pmatrix} \epsilon_{rqq} \\ \epsilon_{rqp} \\ \epsilon_{rqr} \end{pmatrix} & \begin{pmatrix} \epsilon_{rpq} \\ \epsilon_{rpp} \\ \epsilon_{rpr} \end{pmatrix} & \begin{pmatrix} \epsilon_{rrq} \\ \epsilon_{rrp} \\ \epsilon_{rrr} \end{pmatrix} \end{pmatrix}$$

Out[18]//MatrixForm=

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

```
Out[19]= True
```

The rule automatically selects the set of base indices that is associated with the flavor of the indices.

```
In[21]:= DeclareBaseIndices[{x, y, z}, {red, {ρ, θ, ϕ}}]
```

```
In[22]:= MapThread[{#, # /. PermutationSymbolRule[ϵ]} &,
           {{ϵddd[a, c, b], ϵuuu[a, b, c] // ToFlavor[red],
             ϵddd[x, y, z], ϵddd[θ, ϕ, ρ] // ToFlavor[red], ϵddd[red@θ, z, x],
             ϵddd[x, z, y], ϵddd[x, y, y], ϵdud[1, 3, 2]}}] // TableForm
```

Out[22]//TableForm=

ϵ_{acb}	ϵ_{acb}
ϵ^{abc}	ϵ^{abc}
ϵ_{xyz}	1
$\epsilon_{\theta\phi\rho}$	1
$\epsilon_{\theta zx}$	$\epsilon_{\theta zx}$
ϵ_{xzy}	−1
ϵ_{xyy}	0
$\epsilon_{1}{}^{3}{}_{2}$	$\epsilon_{1}{}^{3}{}_{2}$

Restore settings.

```
In[23]:= DeclareBaseIndices @@ oldindices
         DeclareIndexFlavor[oldflavors];
         ClearTensorShortcuts[ϵ, 3]
         Clear[oldflavors, oldindices]
```

PermuteTensorSlots

- `PermuteTensorSlots[`*label, permutation*`][`*expr*`]` will permute all tensors with the specified label and order according to the permutation. The order (number of slots) is determined by the length of the permutation.
- `PermuteTensorSlots[`*label*`][`*expr*`]` will put all the up slots first and down slots last without reordering within the ups and downs.

Permuting slots is not always correct and so this command must be used with care.

The permutation list must be a permutation of `Range[Length[permutation]]`.

See also: `TensorSymmetry`, `IndexChange`.

Examples

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

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

The following permutes the slots of all 4th order tensors T.

```
In[3]:= {Tdddd[a, b, c, d], Tdddd[d, b, c, a], Tuddd[d, b, c, a]}
        % // PermuteTensorSlots[T, {4, 2, 3, 1}]

Out[3]= {T_abcd, T_dbca, T^d_bca}

Out[4]= {T_dbca, T_abcd, T_abc^d}
```

The permutation list must be a permutation of `Range[Length[permutation]]`.

```
In[5]:= Tdd[a, b]
        % // PermuteTensorSlots[T, {2, 1}]
        % // PermuteTensorSlots[T, {2, 1}]
        % // PermuteTensorSlots[T, {2, 2}]
        % // PermuteTensorSlots[T, {2, x}]
        % // PermuteTensorSlots[T, {3, 1}]

Out[5]= T_ab

Out[6]= T_ba

Out[7]= T_ab

        PermuteTensorSlots::permutation : Permutation {2, 2} is invalid.

Out[8]= T_ab

        PermuteTensorSlots::permutation : Permutation {2, x} is invalid.

Out[9]= T_ab

        PermuteTensorSlots::permutation : Permutation {3, 1} is invalid.

Out[10]= T_ab
```

The following statements move all the up slots to the first positions.

```
In[11]:= Tdduu[b, a, d, c]
         % // PermuteTensorSlots[T]
```

$$\text{Out[11]= } T_{b\,a}{}^{d\,c}$$

$$\text{Out[12]= } T^{d\,c}{}_{b\,a}$$

```
In[13]:= Tudud[b, a, d, c]
         % // PermuteTensorSlots[T]
```

$$\text{Out[13]= } T^{b}{}_{a}{}^{d}{}_{c}$$

$$\text{Out[14]= } T^{b\,d}{}_{a\,c}$$

```
In[15]:= ClearTensorShortcuts[{T, 2}, {T, 4}]
```

PushOnto

- `PushOnto[`*arglist, ontolist*`][`*expr*`]` is a form of the Through command that pushes arguments only onto forms given in the ontolist.
- `PushOnto[`*ontolist*`][(`*head*`)[`*args*`]]` pushes args onto forms given in the onto list.

PushOnto is used by the EvaluateSlots routine but is made available to the Tensorial user.

See also: `EvaluateSlots`, `LinearBreakout`, `CircleEvalRule`.

Examples

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

Here is a linear expression of dyads to be evaluated on the vector s. We can use PushOnto to push the arguments onto each direct product. CircleEvalRule does the final evaluation.

```
In[2]:= (a1 u⊗v + a2 u⊗w + a3 v⊗w)[, s]
        % // PushOnto[{CircleTimes[__]}]
        % /. CircleEvalRule

Out[2]= (a1 u⊗v + a2 u⊗w + a3 v⊗w)[Null, s]

Out[3]= a1 (u⊗v)[Null, s] + a2 (u⊗w)[Null, s] + a3 (v⊗w)[Null, s]

Out[4]= a1 u v.s + a2 u w.s + a3 v w.s
```

The following pushes the arguments onto a sum of three functions.

```
In[5]:= (f + g + h)[x, y, z]
        % // PushOnto[{f, g, h}]

Out[5]= (f + g + h)[x, y, z]

Out[6]= f[x, y, z] + g[x, y, z] + h[x, y, z]
```

The following pushes the arguments only onto the functional part of an operator.

```
In[7]:= (1 + 3 x y^2 + h)[x, y, z]
        % // PushOnto[{x, y, z}, {h}]

Out[7]= (1 + h + 3 x y^2)[x, y, z]

Out[8]= 1 + 3 x y^2 + h[x, y, z]
```

The following pushes onto a function and derivatives of the function and then evaluates for a specific function.

```
In[9]:= λ h + Plus @@ Table[(Derivative @@ (2 Part[IdentityMatrix[3], i]))[h], {i, 3}]
        %[x, y, z]
        % // PushOnto[{h, Derivative[_, _, _][h]}]
        % /. h → Function[{x, y, z}, Sin[x] Cos[y] Exp[-z]] // Simplify
```

$$\text{Out[9]= } h\,\lambda + h^{(0,0,2)} + h^{(0,2,0)} + h^{(2,0,0)}$$

$$\text{Out[10]= } (h\,\lambda + h^{(0,0,2)} + h^{(0,2,0)} + h^{(2,0,0)})[x, y, z]$$

$$\text{Out[11]= } \lambda\, h[x, y, z] + h^{(0,0,2)}[x, y, z] + h^{(0,2,0)}[x, y, z] + h^{(2,0,0)}[x, y, z]$$

$$\text{Out[12]= } e^{-z}\,(-1 + \lambda)\,\text{Cos}[y]\,\text{Sin}[x]$$