DeclareBaseIndices

- `DeclareBaseIndices[`*standardbaseindices* : {*index* ..}, *flavoredbases* ... `]` declares the base indices for the underlying linear space and any subsidiary spaces associated with special flavored indices.

The indices should be integers or Symbols.

The standardbaseindices are stored in `BaseIndices` and their length is stored in `NDim`. This is mainly a holdover from Version 3.0

The complete set of base indices is stored in `CompleteBaseIndices`.

`GetBaseIndices` can be used to retrieve the basse indices for any index.

When Tensorial is loaded `BaseIndices` is initialized to `{1, 2, 3}` and `CompleteBaseIndices` is initialized to `{{1, 2, 3}}`.

If you plan to switch between coordinate systems, say Cartesian `{x, y, z}` and spherical `{ρ, θ, ϕ}`, it is better to use integers for the base indices with the flavor indicating the coordinate system. If you picked the base indices to be `{x, y, z}` then even though you used a different flavor for spherical, the base indices would still be, say, `{x, y, z}`. However, you could also use the next feature...

Flavored indices can also be associated with different sets of base indices and different dimensional spaces. This is done with the optional arguments flavoredbases which will consist of entries of the followinng form `{flavor, baseindices}`, for example `{red, {A, B}}`. Any flavors of indices not named in `DeclareBaseIndices` are associated with the standard base indices. Array and sum expansions are done on the associated base indices.

`DeclareBaseIndices` won't evaluate if any flavor names have not been declared with `DeclareIndexFlavor`.

Each `DeclareBaseIndices` statement completely resets all of the base indices.

See also: `GetBaseIndices`, `BaseIndices`, `NDim`, `DeclareIndexFlavor`, `BaseIndexQ`.

Examples

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

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        DeclareIndexFlavor[{red, Red}, {space, SuperStar}, {blue, Blue}]
```

The following statement is automatically evaluated when Tensorial is loaded setting the default dimension and base indices for a tensor system.

```
In[5]:= DeclareBaseIndices[{1, 2, 3}]
        {NDim, BaseIndices}
        CompleteBaseIndices

Out[6]= {3, {1, 2, 3}}

Out[7]= {{1, 2, 3}}
```

The indices must have an Integer or Symbol head.

```
In[8]:= DeclareBaseIndices[{1, 2, f[x]}]
        {NDim, BaseIndices}

Out[8]= DeclareBaseIndices[{1, 2, f[x]}]

Out[9]= {3, {1, 2, 3}}
```

The indices do not have to start at 1.

```
In[10]:= DeclareBaseIndices[{0, 1, 2, 3}]
         {NDim, BaseIndices}

Out[11]= {4, {0, 1, 2, 3}}
```

Symbols can be used

```
In[12]:= DeclareBaseIndices[{t, x, y, z}]
         {NDim, BaseIndices}

Out[13]= {4, {t, x, y, z}}
```

The following statement associates separate sets of base indices with red and space flavored indices.

```
In[14]:= DeclareBaseIndices[{0, 1, 2, 3}, {red, {A, B}}, {space, {1, 2, 3}}]
```

Now red and space flavored indices have their own sets of base indices, but black and blue indices have the standard base indices.

```
In[15]:= {a, red@a, space@a, blue@a}
         GetBaseIndices /@ %

Out[15]= {a, a, a*, a}

Out[16]= {{0, 1, 2, 3}, {A, B}, {1, 2, 3}, {0, 1, 2, 3}}
```

You could also use this feature to preserve the dimension but declare different symbols for the base indices.

```
In[17]:= DeclareBaseIndices[{x, y, z}, {red, {ρ, θ, ϕ}}]

In[18]:= {a, red@a}
         GetBaseIndices /@ %

Out[18]= {a, a}

Out[19]= {{x, y, z}, {ρ, θ, ϕ}}
```

Restore the initial values...

```
In[20]:= DeclareBaseIndices @@ oldindices
         DeclareIndexFlavors @@ oldflavors;
         Clear[oldindices, oldflavors]
```

DeclareIndexFlavor

- `DeclareIndexFlavor[{`*flavorname, flavorform*`} ...]` will add the index flavors to the IndexFlavors list and establish the Format for displaying indices with the given flavor name.

Index flavors are used to distinguish various coordinate systems or reference frames. It is also possible to associated different sets of base indices, and different dimensions, with various index flavors.

Flavored indices are stored internally as `flavorname[index]` but formatted on output according to flavorform.

flavorname can be any symbol used to identify a coordinate system or frame, for example, red, blue, bar, rocket, lab.

flavorform can be a either an RGBColor or a suitable *Mathematica* "annotated name". Tensorial automatically loads the `Graphics`Colors`` package so any named colors can be used and the indices will be formatted in the corresponding colors.

An annotated name is one of the objects listed in Section 3.10.2 "Names of Symbols and Mathematical Objects" in The *Mathematica* Book. Suitable flavorforms might be SuperPlus, SuperMinus, SuperStar, SuperDagger, OverBar, OverTilde, OverHat.

See `ToFlavor` for examples of using flavored indices.

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

Examples

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

The following gives the current set of of index flavors.

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

This declares new index flavors.

```
In[4]:= DeclareIndexFlavor[{red, Red}, {rocket, SuperStar}];
        IndexFlavors // StandardForm

Out[5]//StandardForm=
      {{red, RGBColor[1, 0, 0]}, {rocket, SuperStar}}
```

Flavored indices are then displayed with the designated forms.

```
In[6]:= {Tensor[x, {red[i]}, {Void}], Tensor[x, {rocket[i]}, {Void}]}

Out[6]= {x^i, x^i*}
```

This resets to the original flavors.

```
In[7]:= ClearIndexFlavor[IndexFlavors];
        DeclareIndexFlavor /@ oldflavors;
        Clear[oldflavors]
```

DeclarePatternSymmetries

- `DeclarePatternSymmetries[`*pattern, symmetries*`]` will store the pattern symmetry specification in PatternSymmetries.
- `DeclarePatternSymmetries[{`*pattern, symmetries*`} ... ]` will store multiple pattern symmetries.

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

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

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

The pattern symmetries may be applied by using `SymmetrizePattern[]` on an expression.

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

Examples

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

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

The current pattern symmetries are stored in `PatternSymmetries`, which is initially empty.

```
In[10]:= PatternSymmetries
```

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

The following declares that the dot product on basis vector is symmetric.

```
In[11]:= DeclarePatternSymmetries[ed[a_].ed[b_], {{1, {a, b}}}]
         PatternSymmetries
```

```
Out[12]= {{e_a.e_b, {{1, {a, b}}}}}
```

This will now be automatically applied to an expression with `SymmetrizePattern[]`.

```
In[13]:= 1 / 2 (ed[i].ed[j] + ed[j].ed[i])
         % // SymmetrizePattern[]
```

Out[13]= $\frac{1}{2}\left(e_i.e_j + e_j.e_i\right)$

Out[14]= $e_i.e_j$

Multiple patterns can be declared at once...

```
In[15]:= DeclarePatternSymmetries[
          {Suud[a_, b_, d_] Tudd[c_, e_, f_], {{1, {a, b, c}}, {1, {d, e, f}}}},
          {PartialD[labs][Suud[a_, b_, d_], xu[_]] Tudd[c_, e_, f_] |
            PartialD[labs][Tudd[c_, e_, f_], xu[_]] Suud[a_, b_, d_],
           {{1, {a, b, c}}, {1, {d, e, f}}}},
          {ed[a_].ed[b_], {{1, {a, b}}}},
          {Rdddd[a_, b_, c_, d_], {{1, {{a, b}, {c, d}}}}}]
         PatternSymmetries
```

Out[16]= $\left\{\left\{T^{c\_}{}_{e\_f\_}\,\frac{\partial S^{a\_b\_}{}_{d\_}}{\partial x\_}\;\middle|\; S^{a\_b\_}{}_{d\_}\,\frac{\partial T^{c\_}{}_{e\_f\_}}{\partial x\_}, \{\{1, \{a, b, c\}\}, \{1, \{d, e, f\}\}\}\right\},\right.$
$\left\{e_{a\_}.e_{b\_}, \{\{1, \{a, b\}\}\}\right\}, \left\{R_{a\_b\_c\_d\_}, \{\{1, \{\{a, b\}, \{c, d\}\}\}\}\right\},$
$\left.\left\{S^{a\_b\_}{}_{d\_}\,T^{c\_}{}_{e\_f\_}, \{\{1, \{a, b, c\}\}, \{1, \{d, e, f\}\}\}\right\}\right\}$

and all applied with the same command.

```
In[17]:= {Suud[a, b, d] Tudd[c, e, f] + Suud[b, a, d] Tudd[c, f, e],
          1 / 2 (ed[i].ed[j] + ed[j].ed[i]),
          PartialD[labs][Suud[α, β, δ] Tudd[γ, ϵ, ϕ] + Suud[β, α, δ] Tudd[γ, ϕ, ϵ], xu[a]],
          Rdddd[γ, δ, α, β] + Rdddd[α, β, γ, δ]}
         % // SymmetrizePattern[]
```

Out[17]= $\left\{S^{a\,b}{}_{d}\,T^{c}{}_{e\,f} + S^{b\,a}{}_{d}\,T^{c}{}_{f\,e}, \frac{1}{2}\left(e_i.e_j + e_j.e_i\right),\right.$
$\left.T^{\gamma}{}_{\epsilon\,\phi}\,\frac{\partial S^{\alpha\,\beta}{}_{\delta}}{\partial x^a} + T^{\gamma}{}_{\phi\,\epsilon}\,\frac{\partial S^{\beta\,\alpha}{}_{\delta}}{\partial x^a} + S^{\alpha\,\beta}{}_{\delta}\,\frac{\partial T^{\gamma}{}_{\epsilon\,\phi}}{\partial x^a} + S^{\beta\,\alpha}{}_{\delta}\,\frac{\partial T^{\gamma}{}_{\phi\,\epsilon}}{\partial x^a}, R_{\alpha\beta\gamma\delta} + R_{\gamma\delta\alpha\beta}\right\}$

Out[18]= $\left\{2\,S^{a\,b}{}_{d}\,T^{c}{}_{e\,f}, e_i.e_j, 2\,T^{\gamma}{}_{\epsilon\,\phi}\,\frac{\partial S^{\alpha\,\beta}{}_{\delta}}{\partial x^a} + 2\,S^{\alpha\,\beta}{}_{\delta}\,\frac{\partial T^{\gamma}{}_{\epsilon\,\phi}}{\partial x^a}, 2\,R_{\alpha\beta\gamma\delta}\right\}$

```
In[19]:= (Suud[a, b, d] Tudd[c, e, f] + Suud[b, a, d] Tudd[c, f, e])
           1 / 2 (ed[i].ed[j] + ed[j].ed[i]) // Expand
         % // SymmetrizePattern[]
```

Out[19]= $\frac{1}{2}\,e_i.e_j\,S^{a\,b}{}_{d}\,T^{c}{}_{e\,f} + \frac{1}{2}\,e_j.e_i\,S^{a\,b}{}_{d}\,T^{c}{}_{e\,f} + \frac{1}{2}\,e_i.e_j\,S^{b\,a}{}_{d}\,T^{c}{}_{f\,e} + \frac{1}{2}\,e_j.e_i\,S^{b\,a}{}_{d}\,T^{c}{}_{f\,e}$

Out[20]= $2\,e_i.e_j\,S^{a\,b}{}_{d}\,T^{c}{}_{e\,f}$

Restore the original state.

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

DeclareZeroTensor

- `DeclareZeroTensor[`*label*`]` declares that a tensor of any type with the given label will have zero element values when expanded to an array.

Using a zero tensor allows free indices to balance on both sides of an equation.

Plus and Times are altered so that zero tensors will combine like 0 in sums and products.

Tensor shortcuts must still be defined for a zero tensor if you intend to use them.

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

Examples

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

Save the settings.

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

```
In[3]:= DeclareBaseIndices[{1, 2}]
        DefineTensorShortcuts[{{λ, p}, 1}]
```

Use zero as the label for zero tensors. We could have used any label. (But the capital O letter is protected in *Mathematica*.)

```
In[5]:= DeclareZeroTensor[zero]
```

We still have to define shortcuts.

```
In[6]:= DefineTensorShortcuts[{zero, 1}, {zero, 2}, {zero, 3}]
```

Now the following tensors will expand to zero.

```
In[7]:= zerou[a]
        % // EinsteinArray[]
```

```
Out[7]= zero^a
```

```
Out[8]= {0, 0}
```

```
In[9]:= zerouu[a, b]
        % // EinsteinArray[] // MatrixForm
```

```
Out[9]= zero^(a b)
```

```
Out[10]//MatrixForm=
      (0  0)
      (0  0)
```

```
In[11]:= zerouud[a, b, c]
         % // EinsteinArray[] // MatrixForm
```

$$\text{Out[11]= } \text{zero}^{a\,b}{}_{c}$$

Out[12]//MatrixForm=

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

Zero tensors combine as a 0 in sums and products.

```
In[13]:= HoldForm @@ {f[3, zerod[a]]} /. f → Times
         % // ReleaseHold
```

$$\text{Out[13]= } 3\ \text{zero}_{a}$$

$$\text{Out[14]= } \text{zero}_{a}$$

```
In[15]:= HoldForm @@ {f[λu[a], pu[a], zerod[a]]} /. f → Plus
         % // ReleaseHold
```

$$\text{Out[15]= } \lambda^{a} + p^{a} + \text{zero}_{a}$$

$$\text{Out[16]= } p^{a} + \lambda^{a}$$

The equation for parallel transport of a vector is often written in the following way in the books. But if we try to expand this to an array of equations, Tensorial objects because the free indices do not match. There are no free indices on the right hand side.

```
In[17]:= AbsoluteD[λu[a], u] == 0
         MapAt[ExpandAbsoluteD[{x, δ, g, Γ}, {c, d}], %, 1]
         % // EinsteinArray[] // TableForm
```

$$\text{Out[17]= } \frac{D\,\lambda^{a}}{d\,u} == 0$$

$$\text{Out[18]= } \Gamma^{a}{}_{c\,d}\,\lambda^{c}\,\frac{dx^{d}}{du} + \frac{d\lambda^{a}}{du} == 0$$

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

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

Using a zero tensor provides a balanced equation that can be expanded without fuss.

```
In[20]:= AbsoluteD[λu[a], u] == zerou[a]
         MapAt[ExpandAbsoluteD[{x, δ, g, Γ}, {c, d}], %, 1]
         % // EinsteinArray[] // TableForm
```

$$\text{Out[20]= } \frac{D\,\lambda^{a}}{d\,u} == \text{zero}^{a}$$

$$\text{Out[21]= } \Gamma^{a}{}_{c\,d}\,\lambda^{c}\,\frac{dx^{d}}{du} + \frac{d\lambda^{a}}{du} == \text{zero}^{a}$$

Out[22]//TableForm=

$$\Gamma^{1}{}_{c\,d}\,\lambda^{c}\,\frac{dx^{d}}{du} + \frac{d\lambda^{1}}{du} == 0$$

$$\Gamma^{2}{}_{c\,d}\,\lambda^{c}\,\frac{dx^{d}}{du} + \frac{d\lambda^{2}}{du} == 0$$

```
In[23]:= DeclareBaseIndices @@ oldindices
         Clear[oldindices]
```

DefineTensorShortcuts

- `DefineTensorShortcuts[`*label, order*`]` will define tensor shortcuts for all up/down configurations of a tensor with the given label and order.
- `DefineTensorShortcuts[{`*u, v, w, ...*`}, `*order*`]` will define tensor shortcuts for a number of labels.
- `DefineTensorShortcuts[{{`*u, v, w, ...*`}, `*order*`}..]` will define a number of tensors of different orders.

The shortcut names are derived by appending u's and d's to the label, in the order of the tensor slots.

If the tensor label is `T` and the order is 2, then the shortcut names will be `{Tuu, Tud, Tdu, Tdd}`. `Tdu[i, j]` will be the shortcut for `Tensor[T, {Void, j}, {i, Void}]`.

In the shortcut the indices are specified in the physical order they appear in the tensor slots.

Tensor shortcuts affect only the input of tensors.

Tensorial tensor shortcuts are probably the minimum keystroke method of entering tensors.

You can also write your own direct shortcut definitions, say for a combination of tensors or for a high order tensor in which you don't want all the combinations of up/down indices.

See also: `ClearTensorShortcuts`, `ClearTensorValues`, `SetTensorValueRules`, `SetTensorValues`.

Examples

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

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

This defines all the shortcuts for a coordinate object, the metric tensor and a second order tensor R.

```
In[7]:= DefineTensorShortcuts[x, 1];
        DefineTensorShortcuts[{g, R}, 2]
```

We can now use the shortcuts instead of the longer form.

```
In[9]:= {gdd[i, j], guu[i, j], Rud[i, j], xu[k], xd[k]}
```

```
Out[9]= {g_{i j}, g^{i j}, R^i_j, x^k, x_k}
```

We can obtain a different flavor for the indices by using ToFlavor.

```
In[10]:= {gdd[i, j], guu[i, j], Rud[i, j], xu[k], xd[k]} // ToFlavor[red]
```

```
Out[10]= {g_{i j}, g^{i j}, R^i_j, x^k, x_k}
```

This clears the tensor shortcuts.

```
In[11]:= ClearTensorShortcuts[x, 1];
         ClearTensorShortcuts[{g, R}, 2]
```

They are no longer effective.

```
In[13]:= {gdd[i, j], guu[i, j], Rud[i, j], xu[k], xd[k]}

Out[13]= {gdd[i, j], guu[i, j], Rud[i, j], xu[k], xd[k]}
```

Multiple shortcuts of different orders can be defined in one statement. The following reestablishes the definitions.

```
In[14]:= DefineTensorShortcuts[{x, 1}, {{g, R}, 2}]
         {gdd[i, j], guu[i, j], Rud[i, j], xu[k], xd[k]} // ToFlavor[rocket]

Out[15]= {g_{i* j*}, g^{i* j*}, R^{i*}_{j*}, x^{k*}, x_{k*}}
```

It is then easy to enter expressions and perform manipulations. Here are various examples using tensor shortcuts.

```
In[16]:= gdd[i, j] xu[j] // ToFlavor[red]
         % // MetricSimplify[g]
         (guu[i, j] // ToFlavor[red]) %
         % // MetricSimplify[g]

Out[16]= g_{i j} x^j

Out[17]= x_i

Out[18]= g^{i j} x_i

Out[19]= x^j
```

This defines shortcuts for a vector, v.

```
In[20]:= DefineTensorShortcuts[v, 1]

In[21]:= vu[j]
         % // EinsteinArray[]

Out[21]= v^j

Out[22]= {v^1, v^2}
```

Let R represent a rotation tensor that will operate on vectors. We have already defined the shortcuts. We define values for conversion from a black frame to a red frame.

```
In[23]:= SetTensorValueRules[Rud[red@i, j], ( Cos[θ]  -Sin[θ] )
                                            ( Sin[θ]   Cos[θ] )]

In[24]:= TensorValueRules[R]

Out[24]= {R^1_1 → Cos[θ], R^1_2 → -Sin[θ], R^2_1 → Sin[θ], R^2_2 → Cos[θ]}
```

Notice that the indices in the shortcut fill the tensor slots in order. We can now write our transformation equations, expand them and substitute the values of the rotation matrix.

```
In[25]:= vu[red@i] == Rud[red@i, j] vu[j]
         % // EinsteinSum[]
         % // EinsteinArray[] // TableForm
         % /. TensorValueRules[R] // TableForm
```

$$\text{Out[25]=}\quad v^{i} == R^{i}{}_{j}\, v^{j}$$

$$\text{Out[26]=}\quad v^{i} == R^{i}{}_{1}\, v^{1} + R^{i}{}_{2}\, v^{2}$$

Out[27]//TableForm=

$$v^{1} == R^{1}{}_{1}\, v^{1} + R^{1}{}_{2}\, v^{2}$$
$$v^{2} == R^{2}{}_{1}\, v^{1} + R^{2}{}_{2}\, v^{2}$$

Out[28]//TableForm=

$$v^{1} == \text{Cos}[\theta]\, v^{1} - \text{Sin}[\theta]\, v^{2}$$
$$v^{2} == \text{Sin}[\theta]\, v^{1} + \text{Cos}[\theta]\, v^{2}$$

```
In[29]:= ClearTensorValues[Rud[red@i, j]]
         ClearTensorShortcuts[{{x, v}, 1}, {{g, R}, 2}]
```

It is also possible to define tensor labels that have 'u' and/or 'd' as trailing characters and these will not interfere with labels that don't have the trailing characters. The definitions are different because they have a different number of index arguments.

```
In[31]:= DefineTensorShortcuts[{{T, Tu, Tud}, 1}, {{T, Tu}, 2}]
```

```
In[32]:= {Tu[i], Tuu[i], Tud[i], Tudu[i], Tuu[i, j], Tuud[i, j]}
```

$$\text{Out[32]=}\quad \left\{T^{i},\ Tu^{i},\ Tu_{i},\ Tud^{i},\ T^{i\,j},\ Tu^{i}{}_{j}\right\}$$

```
In[33]:= ClearTensorShortcuts[{{T, Tu, Tud}, 1}, {{T, Tu}, 2}]
```

Restore the original state.

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

DotOperate

- `DotOperate[`*position, operation*`][`*term*`]` will carry out the operation on the arrays at position and position + 1 in a dot mode tensor expression.

The routines in the Arrays section of Tensorial Help facilitate the conversion of tensor equations to vector-matrix-array equations. In Tensorial this is called operating in *dot mode*. This can be used for didactic purposes and is sometimes faster because *Mathematica* array operations are more efficient than tensor summations. See `Arrays & Tensors` in the Examples section for an extended discussion.

Operation will generally be a Function for the dot product of two arrays that may transpose levels in each of the arrays, and contract a number of levels in the resulting array. It will be of the form

```
Function[{A, B},
 ContractArray[Transpose[A, transpositions].Transpose[B, transpositions], contractions]]
```

You may also need to Transpose the final array to obtain the desired order for the free indices.

In each factor of a dot product it is always the *sort order* of the indices that determines the levels of the slots in the array. When two arrays `A.B` are contracted they are always contracted on the lowest level of `A` and the highest level of `B`. Hence they may have to be transposed to satisfy this condition.

The final result is always wrapped in `MatrixForm`.

See also: `DotTensorFactors`, `ExpandDotArray`, `ContractArray`, `ToArrayValues`, `ArrayExpansion`, `EinsteinArray`.

Examples

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

Save the old settings.

```
In[2]:= oldindices = CompleteBaseIndices;
        DeclareBaseIndices[{1, 2, 3}]
```

```
In[4]:= DefineTensorShortcuts[{{e, f, R}, 1}, {{R, S, T}, 2}, {{R, S}, 3}]
```

The following is a regular tensor equation expanded to component form.

```
In[5]:= Rdd[a, c] == Tdu[a, b] Sdd[b, c]
        MatrixForm[ToArrayValues[][#]] & /@ %
```

```
Out[5]= R_{a c} == S_{b c} T_a^b
```

$$\text{Out[6]=} \begin{pmatrix} R_{1\,1} & R_{1\,2} & R_{1\,3} \\ R_{2\,1} & R_{2\,2} & R_{2\,3} \\ R_{3\,1} & R_{3\,2} & R_{3\,3} \end{pmatrix} == \begin{pmatrix} S_{1\,1}\,T_1^{\,1} + S_{2\,1}\,T_1^{\,2} + S_{3\,1}\,T_1^{\,3} & S_{1\,2}\,T_1^{\,1} + S_{2\,2}\,T_1^{\,2} + S_{3\,2}\,T_1^{\,3} & S_{1\,3}\,T_1^{\,1} + S_{2\,3}\,T_1^{\,2} + S_{3\,3}\,T_1^{\,3} \\ S_{1\,1}\,T_2^{\,1} + S_{2\,1}\,T_2^{\,2} + S_{3\,1}\,T_2^{\,3} & S_{1\,2}\,T_2^{\,1} + S_{2\,2}\,T_2^{\,2} + S_{3\,2}\,T_2^{\,3} & S_{1\,3}\,T_2^{\,1} + S_{2\,3}\,T_2^{\,2} + S_{3\,3}\,T_2^{\,3} \\ S_{1\,1}\,T_3^{\,1} + S_{2\,1}\,T_3^{\,2} + S_{3\,1}\,T_3^{\,3} & S_{1\,2}\,T_3^{\,1} + S_{2\,2}\,T_3^{\,2} + S_{3\,2}\,T_3^{\,3} & S_{1\,3}\,T_3^{\,1} + S_{2\,3}\,T_3^{\,2} + S_{3\,3}\,T_3^{\,3} \end{pmatrix}$$

In the following the rhs is converted to dot mode. We can now show the individual matrices and if we have many factors the array multiplication might actually be faster.

```
In[7]:= Print["A tensor equation"]
        Rdd[a, c] == Tdu[a, b] Sdd[b, c]
        Print["Converted to dot mode"]
        MapAt[DotTensorFactors[{2, 1}], %%, 2]
        Print["Expanding each of the tensors"]
        %% // ExpandDotArray[Tensor[_, ___]]
        Print["Carrying out the Dot product on the rhs"]
        MapAt[DotOperate[1, Dot], %%, 2]
```

A tensor equation

$$Out[8]= R_{a\,c} == S_{b\,c}\, T_{a}^{\ b}$$

Converted to dot mode

$$Out[10]= R_{a\,c} == T_{a}^{\ b} . S_{b\,c}$$

Expanding each of the tensors

$$Out[12]= \begin{pmatrix} R_{1\,1} & R_{1\,2} & R_{1\,3} \\ R_{2\,1} & R_{2\,2} & R_{2\,3} \\ R_{3\,1} & R_{3\,2} & R_{3\,3} \end{pmatrix} == \begin{pmatrix} T_{1}^{\ 1} & T_{1}^{\ 2} & T_{1}^{\ 3} \\ T_{2}^{\ 1} & T_{2}^{\ 2} & T_{2}^{\ 3} \\ T_{3}^{\ 1} & T_{3}^{\ 2} & T_{3}^{\ 3} \end{pmatrix} . \begin{pmatrix} S_{1\,1} & S_{1\,2} & S_{1\,3} \\ S_{2\,1} & S_{2\,2} & S_{2\,3} \\ S_{3\,1} & S_{3\,2} & S_{3\,3} \end{pmatrix}$$

Carrying out the Dot product on the rhs

$$Out[14]= \begin{pmatrix} R_{1\,1} & R_{1\,2} & R_{1\,3} \\ R_{2\,1} & R_{2\,2} & R_{2\,3} \\ R_{3\,1} & R_{3\,2} & R_{3\,3} \end{pmatrix} == \begin{pmatrix} S_{1\,1} T_{1}^{\ 1} + S_{2\,1} T_{1}^{\ 2} + S_{3\,1} T_{1}^{\ 3} & S_{1\,2} T_{1}^{\ 1} + S_{2\,2} T_{1}^{\ 2} + S_{3\,2} T_{1}^{\ 3} & S_{1\,3} T_{1}^{\ 1} + S_{2\,3} T_{1}^{\ 2} + S_{3\,3} T_{1}^{\ 3} \\ S_{1\,1} T_{2}^{\ 1} + S_{2\,1} T_{2}^{\ 2} + S_{3\,1} T_{2}^{\ 3} & S_{1\,2} T_{2}^{\ 1} + S_{2\,2} T_{2}^{\ 2} + S_{3\,2} T_{2}^{\ 3} & S_{1\,3} T_{2}^{\ 1} + S_{2\,3} T_{2}^{\ 2} + S_{3\,3} T_{2}^{\ 3} \\ S_{1\,1} T_{3}^{\ 1} + S_{2\,1} T_{3}^{\ 2} + S_{3\,1} T_{3}^{\ 3} & S_{1\,2} T_{3}^{\ 1} + S_{2\,2} T_{3}^{\ 2} + S_{3\,2} T_{3}^{\ 3} & S_{1\,3} T_{3}^{\ 1} + S_{2\,3} T_{3}^{\ 2} + S_{3\,3} T_{3}^{\ 3} \end{pmatrix}$$

Here, since the indexes were in the order $a\ b\ b\ c \to a\ c$, we were able to do a simple Dot operation.

The following is the same case, but with the tensors in the opposite order. The indicies are in the order $b\ c\ a\ b$. We must transpose both matrices to obtain $c\ b\ b\ a \to c\ a$. We must then transpose the final result to obtain the free index order $a\ c$.

```
In[15]:= Print["A tensor equation"]
         Rdd[a, c] == Tdu[a, b] Sdd[b, c]
         Print["Converted to dot mode"]
         MapAt[DotTensorFactors[{1, 2}], %%, 2]
         Print["Expanding each of the tensors"]
         %% // ExpandDotArray[Tensor[_, __]]
         Print["Carrying out the Dot product on the rhs"]
         MapAt[DotOperate[1, Function[{A, B}, Transpose[Transpose[A].Transpose[B]]]], %%, 2]
```

A tensor equation

$$\text{Out[16]=}\quad R_{a\,c} == S_{b\,c}\,T_{a}^{\ b}$$

Converted to dot mode

$$\text{Out[18]=}\quad R_{a\,c} == S_{b\,c}.T_{a}^{\ b}$$

Expanding each of the tensors

$$\text{Out[20]=}\quad \begin{pmatrix} R_{1\,1} & R_{1\,2} & R_{1\,3} \\ R_{2\,1} & R_{2\,2} & R_{2\,3} \\ R_{3\,1} & R_{3\,2} & R_{3\,3} \end{pmatrix} == \begin{pmatrix} S_{1\,1} & S_{1\,2} & S_{1\,3} \\ S_{2\,1} & S_{2\,2} & S_{2\,3} \\ S_{3\,1} & S_{3\,2} & S_{3\,3} \end{pmatrix} . \begin{pmatrix} T_{1}^{\ 1} & T_{1}^{\ 2} & T_{1}^{\ 3} \\ T_{2}^{\ 1} & T_{2}^{\ 2} & T_{2}^{\ 3} \\ T_{3}^{\ 1} & T_{3}^{\ 2} & T_{3}^{\ 3} \end{pmatrix}$$

Carrying out the Dot product on the rhs

$$\text{Out[22]=}\quad \begin{pmatrix} R_{1\,1} & R_{1\,2} & R_{1\,3} \\ R_{2\,1} & R_{2\,2} & R_{2\,3} \\ R_{3\,1} & R_{3\,2} & R_{3\,3} \end{pmatrix} == \begin{pmatrix} S_{1\,1}\,T_{1}^{\ 1} + S_{2\,1}\,T_{1}^{\ 2} + S_{3\,1}\,T_{1}^{\ 3} & S_{1\,2}\,T_{1}^{\ 1} + S_{2\,2}\,T_{1}^{\ 2} + S_{3\,2}\,T_{1}^{\ 3} & S_{1\,3}\,T_{1}^{\ 1} + S_{2\,3}\,T_{1}^{\ 2} + S_{3\,3}\,T_{1}^{\ 3} \\ S_{1\,1}\,T_{2}^{\ 1} + S_{2\,1}\,T_{2}^{\ 2} + S_{3\,1}\,T_{2}^{\ 3} & S_{1\,2}\,T_{2}^{\ 1} + S_{2\,2}\,T_{2}^{\ 2} + S_{3\,2}\,T_{2}^{\ 3} & S_{1\,3}\,T_{2}^{\ 1} + S_{2\,3}\,T_{2}^{\ 2} + S_{3\,3}\,T_{2}^{\ 3} \\ S_{1\,1}\,T_{3}^{\ 1} + S_{2\,1}\,T_{3}^{\ 2} + S_{3\,1}\,T_{3}^{\ 3} & S_{1\,2}\,T_{3}^{\ 1} + S_{2\,2}\,T_{3}^{\ 2} + S_{3\,2}\,T_{3}^{\ 3} & S_{1\,3}\,T_{3}^{\ 1} + S_{2\,3}\,T_{3}^{\ 2} + S_{3\,3}\,T_{3}^{\ 3} \end{pmatrix}$$

The following case requires a contraction of the resulting matrix. In the dot mode the indices are in the order `a b a b c` we must transpose the S matrix to obtain $\texttt{b a a b c} \to \texttt{b b c}$. This must then be contracted on the first two levels to give `c`.

```
In[23]:= Print["A tensor equation"]
         ed[c] == Ruud[a, b, c] Sdd[a, b]
         Print["Converted to dot mode"]
         MapAt[DotTensorFactors[{2, 1}], %%, 2]
         Print["Expanding each of the tensors"]
         %% // ExpandDotArray[Tensor[_, ___]]
         Print["Carrying out the Dot operation on the rhs"]
         MapAt[
          DotOperate[1, Function[{A, B}, ContractArray[Transpose[A].B, {1, 2}]]], %%, 2]
```

A tensor equation

Out[24]= $e_c == R^{a\,b}{}_c\, S_{a\,b}$

Converted to dot mode

Out[26]= $e_c == S_{a\,b}.R^{a\,b}{}_c$

Expanding each of the tensors

$$\text{Out[28]=}\quad \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} == \begin{pmatrix} S_{1\,1} & S_{1\,2} & S_{1\,3} \\ S_{2\,1} & S_{2\,2} & S_{2\,3} \\ S_{3\,1} & S_{3\,2} & S_{3\,3} \end{pmatrix} . \begin{pmatrix} \begin{pmatrix} R^{1\,1}{}_1 \\ R^{1\,1}{}_2 \\ R^{1\,1}{}_3 \end{pmatrix} & \begin{pmatrix} R^{1\,2}{}_1 \\ R^{1\,2}{}_2 \\ R^{1\,2}{}_3 \end{pmatrix} & \begin{pmatrix} R^{1\,3}{}_1 \\ R^{1\,3}{}_2 \\ R^{1\,3}{}_3 \end{pmatrix} \\ \begin{pmatrix} R^{2\,1}{}_1 \\ R^{2\,1}{}_2 \\ R^{2\,1}{}_3 \end{pmatrix} & \begin{pmatrix} R^{2\,2}{}_1 \\ R^{2\,2}{}_2 \\ R^{2\,2}{}_3 \end{pmatrix} & \begin{pmatrix} R^{2\,3}{}_1 \\ R^{2\,3}{}_2 \\ R^{2\,3}{}_3 \end{pmatrix} \\ \begin{pmatrix} R^{3\,1}{}_1 \\ R^{3\,1}{}_2 \\ R^{3\,1}{}_3 \end{pmatrix} & \begin{pmatrix} R^{3\,2}{}_1 \\ R^{3\,2}{}_2 \\ R^{3\,2}{}_3 \end{pmatrix} & \begin{pmatrix} R^{3\,3}{}_1 \\ R^{3\,3}{}_2 \\ R^{3\,3}{}_3 \end{pmatrix} \end{pmatrix}$$

Carrying out the Dot operation on the rhs

$$\text{Out[30]=}\quad \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} == \begin{pmatrix} R^{1\,1}{}_1\, S_{1\,1} + R^{1\,2}{}_1\, S_{1\,2} + R^{1\,3}{}_1\, S_{1\,3} + R^{2\,1}{}_1\, S_{2\,1} + R^{2\,2}{}_1\, S_{2\,2} + R^{2\,3}{}_1\, S_{2\,3} + R^{3\,1}{}_1\, S_{3\,1} + R^{3\,2}{}_1\, S_{3\,2} + R^{3\,3}{}_1\, S_{3\,3} \\ R^{1\,1}{}_2\, S_{1\,1} + R^{1\,2}{}_2\, S_{1\,2} + R^{1\,3}{}_2\, S_{1\,3} + R^{2\,1}{}_2\, S_{2\,1} + R^{2\,2}{}_2\, S_{2\,2} + R^{2\,3}{}_2\, S_{2\,3} + R^{3\,1}{}_2\, S_{3\,1} + R^{3\,2}{}_2\, S_{3\,2} + R^{3\,3}{}_2\, S_{3\,3} \\ R^{1\,1}{}_3\, S_{1\,1} + R^{1\,2}{}_3\, S_{1\,2} + R^{1\,3}{}_3\, S_{1\,3} + R^{2\,1}{}_3\, S_{2\,1} + R^{2\,2}{}_3\, S_{2\,2} + R^{2\,3}{}_3\, S_{2\,3} + R^{3\,1}{}_3\, S_{3\,1} + R^{3\,2}{}_3\, S_{3\,2} + R^{3\,3}{}_3\, S_{3\,3} \end{pmatrix}$$

The following example is continued from a `DotTensorFactors` Example. We have to again perform a Dot product and then contract the resulting array. The indices go `a b a b c`, transpose first matrix, `b a a b c`, dot product, `b b c`, contract, `c`.

```
In[31]:= Rd[c] == 3 eu[a] fu[b] Sddd[a, b, c]
         % // MapLevelParts[DotTensorFactors[{{1, 2}, 3}], {2, {2, 3, 4}}]
         % // ExpandDotArray[Tensor[S, __] | eu[_] fu[_]]
         MapAt[
          DotOperate[1, Function[{A, B}, ContractArray[Transpose[A].B, {1, 2}]]], %, {2, 2}]
```

$$\text{Out[31]=}\quad R_c == 3\, e^a\, f^b\, S_{abc}$$

$$\text{Out[32]=}\quad R_c == 3\, (e^a\, f^b).S_{abc}$$

$$\text{Out[33]=}\quad R_c == 3 \begin{pmatrix} e^1 f^1 & e^1 f^2 & e^1 f^3 \\ e^2 f^1 & e^2 f^2 & e^2 f^3 \\ e^3 f^1 & e^3 f^2 & e^3 f^3 \end{pmatrix} . \begin{pmatrix} \begin{pmatrix} S_{111} \\ S_{112} \\ S_{113} \end{pmatrix} & \begin{pmatrix} S_{121} \\ S_{122} \\ S_{123} \end{pmatrix} & \begin{pmatrix} S_{131} \\ S_{132} \\ S_{133} \end{pmatrix} \\ \begin{pmatrix} S_{211} \\ S_{212} \\ S_{213} \end{pmatrix} & \begin{pmatrix} S_{221} \\ S_{222} \\ S_{223} \end{pmatrix} & \begin{pmatrix} S_{231} \\ S_{232} \\ S_{233} \end{pmatrix} \\ \begin{pmatrix} S_{311} \\ S_{312} \\ S_{313} \end{pmatrix} & \begin{pmatrix} S_{321} \\ S_{322} \\ S_{323} \end{pmatrix} & \begin{pmatrix} S_{331} \\ S_{332} \\ S_{333} \end{pmatrix} \end{pmatrix}$$

$$\text{Out[34]=}\quad R_c == 3 \begin{pmatrix} e^1 f^1 S_{111} + e^1 f^2 S_{121} + e^1 f^3 S_{131} + e^2 f^1 S_{211} + e^2 f^2 S_{221} + e^2 f^3 S_{231} + e^3 f^1 S_{311} + e^3 f^2 \\ e^1 f^1 S_{112} + e^1 f^2 S_{122} + e^1 f^3 S_{132} + e^2 f^1 S_{212} + e^2 f^2 S_{222} + e^2 f^3 S_{232} + e^3 f^1 S_{312} + e^3 f^2 \\ e^1 f^1 S_{113} + e^1 f^2 S_{123} + e^1 f^3 S_{133} + e^2 f^1 S_{213} + e^2 f^2 S_{223} + e^2 f^3 S_{233} + e^3 f^1 S_{313} + e^3 f^2 \end{pmatrix}$$

Restore settings.

```
In[35]:= ClearTensorShortcuts[{{e, f, R}, 1}, {{R, S, T}, 2}, {{R, S}, 3}]
```

```
In[36]:= DeclareBaseIndices @@ oldindices
         Clear[oldindices]
```

DotTensorFactors

- `DotTensorFactors[`*order*`][`*term*`]` will convert a product of tensor factors to a Dot product and put them in the order specified.

The routines in the Arrays section of Tensorial Help facilitate the conversion of tensor equations to vector-matrix-array equations. In Tensorial this is called operating in *dot mode*. This can be used for didactic purposes and is sometimes faster because *Mathematica* array operations are more efficient than tensor summations. See `Arrays & Tensors` in the Examples section for an extended discussion.

order is a list of Parts of the term. Tensor factors may be grouped into one factor in the Dot product using a list of the part numbers. Example:

`T1 T2 T3 T4 // DotTensorFactors[{{2, 3}, 4, 1}]` → `(T2 T3).T4.T1`.

`DotTensorFactors` must be applied individually to each term in a tensor equation because each term may require its own order. If the term contains constant factors it should be mapped onto just the tensor factors using `MapLevelParts`.

When Tensorial expands arrays it is always the sort order of the indices, not the slot position, that determines the levels in the arrays. The lowest sort order index corresponds to the highest array level.

See also: `ExpandDotArray`, `DotOperate`, `ToArrayValues`, `ArrayExpansion`, `EinsteinArray`.

Examples

See also the more extended examples in `ExpandDotArray`.

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

Save the old settings.

```
In[2]:= oldindices = CompleteBaseIndices;
        DeclareBaseIndices[{1, 2, 3}]
```

```
In[4]:= DefineTensorShortcuts[{{e, f, R}, 1}, {{R, S, T}, 2}, {S, 3}]
```

In the following the rhs is converted to dot mode. With a normal tensor expression the `b` index does not line up properly because *Mathematica* sorts S before T. By reordering the factors and putting them in a Dot product we obtain a proper matrix equation.

```
In[5]:= Rdd[a, c] == Tdu[a, b] Sdd[b, c]
        MapAt[DotTensorFactors[{2, 1}], %, 2]
```

```
Out[5]= R_a c == S_b c T_a^b
```

```
Out[6]= R_a c == T_a^b.S_b c
```

The important thing here is that using sort order for the indices in each factor, and lining up the indices for the term we obtain `a b b c`, with the `b`'s adjacent. We also should keep an eye on the lhs and see if those indices are in the order that we want.

In the following we have a constant in the term on the rhs. We must map to only the tensor factors. The `e` and `f` tensors must be grouped together to make a single array.

```
In[7]:= Rd[c] == 3 eu[a] fu[b] Sddd[a, b, c]
        % // MapLevelParts[DotTensorFactors[{{1, 2}, 3}], {2, {2, 3, 4}}]
```

$$\text{Out[7]= } R_c == 3\, e^a\, f^b\, S_{a\,b\,c}$$

$$\text{Out[8]= } R_c == 3\, (e^a\, f^b).S_{a\,b\,c}$$

The DotOperate Help Examples will show how to evaluate this using arrays.

Restore settings.

```
In[9]:= ClearTensorShortcuts[{{e, f, R}, 1}, {{R, S, T}, 2}, {S, 3}]
```

```
In[10]:= DeclareBaseIndices @@ oldindices
         Clear[oldindices]
```