EinsteinArgument

- `EinsteinArgument[x][expr]` will expand coordinate tensors with the label x, or any single index tensor, when used as a single argument of functions.
- `EinsteinArgument[x, f][expr]` will expand only on function heads f.

EinsteinArgument and various features of Tensorial allow calculation with functional notations such as `f[xu[a]]` and various expressions that are used in textbooks. However, it may often be easier to set tensor values and put the expressions in the tensor values.

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

Examples

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

Save the settings and declare base indices and flavors.

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

In[7]:= DefineTensorShortcuts[{{x}, 1}, {δ, 2}]
        DeclareBaseIndices[{1, 2, 3}]
        SetTensorValues[δud[i, j], IdentityMatrix[NDim]]
        SetTensorValues[δud[i, j] // ToFlavor[red], IdentityMatrix[NDim]]
        labs = {x, δ, g, Γ};
```

Define a function f of three coordinates. Leave another function g undefined.

```
In[12]:= f[x_, y_, z_] := x y + y^2 z
```

Following conventional usage we can write...

```
In[13]:= Print["A function of coordinates"]
         g[xu[red@a]]
         Print["Expanded using EinsteinArgument"]
         %% // EinsteinArgument[x]
         Print["Changing g to the defined
            function f automatically evaluates to the expression"]
         %% /.
          g →
           f
```

A function of coordinates

Out[14]= $g[x^a]$

Expanded using EinsteinArgument

Out[16]= $g[x^1, x^2, x^3]$

Changing g to the defined function f automatically evaluates to the expression

Out[18]= $x^1 x^2 + (x^2)^2 x^3$

We can take partial derivatives of functions with coordinate arguments if we use the expanded form with respect to a coordinate.

```
In[19]:= Print["A function of coordinates"]
         g[xu[a]]
         Print["Partial derivative with respect to a coordinate"]
         PartialD[labs][%%, xu[b]]
         Print["Expanding the arguments causes evaluation"]
         %% // EinsteinArgument[x]
         Print["Substituting the defined function and expanding the array"]
         %% /. g → f
         % // EinsteinArray[]
```

A function of coordinates

Out[20]= $g[x^a]$

Partial derivative with respect to a coordinate

Out[22]= $\frac{\partial g[x^a]}{\partial x^b}$

Expanding the arguments causes evaluation

Out[24]= $\frac{\partial x^3}{\partial x^b} g^{(0,0,1)}[x^1, x^2, x^3] + \frac{\partial x^2}{\partial x^b} g^{(0,1,0)}[x^1, x^2, x^3] + \frac{\partial x^1}{\partial x^b} g^{(1,0,0)}[x^1, x^2, x^3]$

Substituting the defined function and expanding the array

Out[26]= $x^2 \frac{\partial x^1}{\partial x^b} + (x^1 + 2 x^2 x^3) \frac{\partial x^2}{\partial x^b} + (x^2)^2 \frac{\partial x^3}{\partial x^b}$

Out[27]= $\left\{x^2, x^1 + 2 x^2 x^3, (x^2)^2\right\}$

To use other derivative constructions it will be necessary to use a nested Tensor to control evaluation.

```
In[28]:= Print["Total derivative of a function of the coordinates"]
         TotalD[Tensor@g[xu[a]], t]
         Print["Expand the arguments and the total derivative in terms of coordinates"]
         %% // EinsteinArgument[x];
         % // ExpandTotalD[labs, b]
         Print["Do an Einstein sum and substitute the defined function f for g"]
         %% // EinsteinSum[]
         % /. g → f
         Print["Unnest the tensor for final evaluation"]
         %% // UnnestTensor
```

Total derivative of a function of the coordinates

$$\text{Out[29]=}\quad \frac{dg[x^a]}{dt}$$

Expand the arguments and the total derivative in terms of coordinates

$$\text{Out[32]=}\quad \frac{dx^b}{dt}\,\frac{\partial g[x^1,\,x^2,\,x^3]}{\partial x^b}$$

Do an Einstein sum and substitute the defined function f for g

$$\text{Out[34]=}\quad \frac{dx^1}{dt}\,\frac{\partial g[x^1,\,x^2,\,x^3]}{\partial x^1}+\frac{dx^2}{dt}\,\frac{\partial g[x^1,\,x^2,\,x^3]}{\partial x^2}+\frac{dx^3}{dt}\,\frac{\partial g[x^1,\,x^2,\,x^3]}{\partial x^3}$$

$$\text{Out[35]=}\quad \frac{dx^1}{dt}\,\frac{\partial x^1\,x^2+(x^2)^2\,x^3}{\partial x^1}+\frac{dx^2}{dt}\,\frac{\partial x^1\,x^2+(x^2)^2\,x^3}{\partial x^2}+\frac{dx^3}{dt}\,\frac{\partial x^1\,x^2+(x^2)^2\,x^3}{\partial x^3}$$

Unnest the tensor for final evaluation

$$\text{Out[37]=}\quad x^2\,\frac{dx^1}{dt}+(x^1+2\,x^2\,x^3)\,\frac{dx^2}{dt}+(x^2)^2\,\frac{dx^3}{dt}$$

For a coordinate transformation let black coordinates be Cartesian coordinates and let the red coordinates be spherical coordinates. Define rules for the coordinate functions.

```
In[38]:= coordinatefunctions =
          Thread[ToArrayValues[][xu[i]] → {Function[{r, θ, ϕ}, r Sin[θ] Cos[ϕ]],
             Function[{r, θ, ϕ}, r Sin[θ] Sin[ϕ]], Function[{r, θ, ϕ}, r Cos[θ]]}]
```

$$\text{Out[38]=}\quad \{x^1 \to \text{Function}[\{r,\,\theta,\,\phi\},\,r\,\text{Sin}[\theta]\,\text{Cos}[\phi]],\ x^2 \to \text{Function}[\{r,\,\theta,\,\phi\},\,r\,\text{Sin}[\theta]\,\text{Sin}[\phi]],\ x^3 \to \text{Function}[\{r,\,\theta,\,\phi\},\,r\,\text{Cos}[\theta]]\}$$

```
In[39]:= Print[
          "Cartesian coordinates (black) as functions of spherical coordinates (red}"]
         xu[a][xu[red@b]]
         Print["Expand the arguments and expand to an array"]
         %% // EinsteinArgument[x]
         % // EinsteinArray[]
         Print["Substitute the coordinate functions and use coordinate symbols"]
         %% /. coordinatefunctions
         % // UseCoordinates[{r, θ, ϕ}, x, red]
```

Cartesian coordinates (black) as functions of spherical coordinates (red}

$$\text{Out[40]= } x^{a}[x^{b}]$$

Expand the arguments and expand to an array

$$\text{Out[42]= } x^{a}[x^{1}, x^{2}, x^{3}]$$

$$\text{Out[43]= } \{x^{1}[x^{1}, x^{2}, x^{3}], x^{2}[x^{1}, x^{2}, x^{3}], x^{3}[x^{1}, x^{2}, x^{3}]\}$$

Substitute the coordinate functions and use coordinate symbols

$$\text{Out[45]= } \{\text{Cos}[x^{3}]\ \text{Sin}[x^{2}]\ x^{1}, \text{Sin}[x^{2}]\ \text{Sin}[x^{3}]\ x^{1}, \text{Cos}[x^{2}]\ x^{1}\}$$

$$\text{Out[46]= } \{r\ \text{Cos}[\phi]\ \text{Sin}[\theta], r\ \text{Sin}[\theta]\ \text{Sin}[\phi], r\ \text{Cos}[\theta]\}$$

The following calculates the Jacobian matrix of the transformation equations.

```
In[47]:= Print["Take the partial derivatives of the coordinate functions"]
         xu[a][xu[red@c]]
         PartialD[labs][%, xu[red@b]]
         Print["Expand to an array and expand the arguments"]
         %% // ToArrayValues[] // MatrixForm
         % // EinsteinArgument[x] // MatrixForm
         Print["Substitute the coordinate functions and use coordinate symbols"]
         %% /. coordinatefunctions // MatrixForm
         % // UseCoordinates[{r, θ, ϕ}, x, red] // MatrixForm
```

Take the partial derivatives of the coordinate functions

$$\text{Out[48]=}\quad x^a[x^c]$$

$$\text{Out[49]=}\quad \frac{\partial x^a[x^c]}{\partial x^b}$$

Expand to an array and expand the arguments

Out[51]//MatrixForm=

$$\begin{pmatrix} \frac{\partial x^1[x^c]}{\partial x^1} & \frac{\partial x^1[x^c]}{\partial x^2} & \frac{\partial x^1[x^c]}{\partial x^3} \\ \frac{\partial x^2[x^c]}{\partial x^1} & \frac{\partial x^2[x^c]}{\partial x^2} & \frac{\partial x^2[x^c]}{\partial x^3} \\ \frac{\partial x^3[x^c]}{\partial x^1} & \frac{\partial x^3[x^c]}{\partial x^2} & \frac{\partial x^3[x^c]}{\partial x^3} \end{pmatrix}$$

Out[52]//MatrixForm=

$$\begin{pmatrix} x^{1(1,0,0)}[x^1, x^2, x^3] & x^{1(0,1,0)}[x^1, x^2, x^3] & x^{1(0,0,1)}[x^1, x^2, x^3] \\ x^{2(1,0,0)}[x^1, x^2, x^3] & x^{2(0,1,0)}[x^1, x^2, x^3] & x^{2(0,0,1)}[x^1, x^2, x^3] \\ x^{3(1,0,0)}[x^1, x^2, x^3] & x^{3(0,1,0)}[x^1, x^2, x^3] & x^{3(0,0,1)}[x^1, x^2, x^3] \end{pmatrix}$$

Substitute the coordinate functions and use coordinate symbols

Out[54]//MatrixForm=

$$\begin{pmatrix} \text{Cos}[x^3]\,\text{Sin}[x^2] & \text{Cos}[x^2]\,\text{Cos}[x^3]\,x^1 & -\text{Sin}[x^2]\,\text{Sin}[x^3]\,x^1 \\ \text{Sin}[x^2]\,\text{Sin}[x^3] & \text{Cos}[x^2]\,\text{Sin}[x^3]\,x^1 & \text{Cos}[x^3]\,\text{Sin}[x^2]\,x^1 \\ \text{Cos}[x^2] & -\text{Sin}[x^2]\,x^1 & 0 \end{pmatrix}$$

Out[55]//MatrixForm=

$$\begin{pmatrix} \text{Cos}[\phi]\,\text{Sin}[\theta] & r\,\text{Cos}[\theta]\,\text{Cos}[\phi] & -r\,\text{Sin}[\theta]\,\text{Sin}[\phi] \\ \text{Sin}[\theta]\,\text{Sin}[\phi] & r\,\text{Cos}[\theta]\,\text{Sin}[\phi] & r\,\text{Cos}[\phi]\,\text{Sin}[\theta] \\ \text{Cos}[\theta] & -r\,\text{Sin}[\theta] & 0 \end{pmatrix}$$

However, it is easier to perform the calculation by setting tensor values.

```
In[56]:= SetTensorValueRules[xu[a], {r Sin[θ] Cos[ϕ], r Sin[θ] Sin[ϕ], r Cos[θ]}]
         SetTensorValueRules[xu[red@a], {r, θ, ϕ}]
         PartialD[labs][xu[a], xu[red@b]]
         % // EinsteinArray[] // MatrixForm
         % // ToArrayValues[] // MatrixForm
         ClearTensorValues[{xu[a], xu[red@a]}]
```

Out[58]=
$$\frac{\partial x^a}{\partial x^b}$$

Out[59]//MatrixForm=
$$\begin{pmatrix} \frac{\partial x^1}{\partial x^1} & \frac{\partial x^1}{\partial x^2} & \frac{\partial x^1}{\partial x^3} \\ \frac{\partial x^2}{\partial x^1} & \frac{\partial x^2}{\partial x^2} & \frac{\partial x^2}{\partial x^3} \\ \frac{\partial x^3}{\partial x^1} & \frac{\partial x^3}{\partial x^2} & \frac{\partial x^3}{\partial x^3} \end{pmatrix}$$

Out[60]//MatrixForm=
$$\begin{pmatrix} \text{Cos}[\phi]\ \text{Sin}[\theta] & r\ \text{Cos}[\theta]\ \text{Cos}[\phi] & -r\ \text{Sin}[\theta]\ \text{Sin}[\phi] \\ \text{Sin}[\theta]\ \text{Sin}[\phi] & r\ \text{Cos}[\theta]\ \text{Sin}[\phi] & r\ \text{Cos}[\phi]\ \text{Sin}[\theta] \\ \text{Cos}[\theta] & -r\ \text{Sin}[\theta] & 0 \end{pmatrix}$$

Restore the initial values...

```
In[62]:= ClearTensorValues[{δud[i, j], δud[i, j] // ToFlavor[red]}]
         ClearTensorShortcuts[{{x}, 1}, {δ, 2}]

In[64]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors, coordinatefunctions]
```

EinsteinArray

■ `EinsteinArray[`*base : Automatic*`][expr]` will form an array on the free indices in the expression.

The expansion will be done in the natural sort order of the raw free indices with the first sorted index at the highest level.

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

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

Flavored indices as well as plain indices are automatically expanded.

See also: `DeclareBaseIndices`, `SumExpansion`, `EinsteinSum`, `ArrayExpansion`, `ToArrayValues`.

Examples

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

Save the settings and declare base indices and flavors.

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

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

The following expands vectors into their components...

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

Out[8]= x^i

Out[9]= {x^1, x^2, x^3}

In[10]:= xu[i] + yu[i]
         % // EinsteinArray[]

Out[10]= x^i + y^i

Out[11]= {x^1 + y^1, x^2 + y^2, x^3 + y^3}
```

An expansion can be done over a subset of the base indices.

```
In[12]:= xu[i] + yu[i]
         % // EinsteinArray[{1, 3}]

Out[12]= x^i + y^i

Out[13]= {x^1 + y^1, x^3 + y^3}
```

If the free indices do not match in all terms, an error message is issued and the operation is aborted.

```
In[14]:= xu[i] + yu[j]
         % // EinsteinArray[]

Out[14]= x^i + y^j

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

Out[15]= $Aborted
```

The following expands a second order tensor into an array.

```
In[16]:= Suu[i, j] // ToFlavor[red]
         % // EinsteinArray[] // MatrixForm

Out[16]= S^(i j)

Out[17]//MatrixForm=
         ( 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) )
```

The expansion occurs on the sort order of the indices. To obtain other expansion orders use `ArrayExpansion` or change the indices in the expression. Changing the sort order of the indices...

```
In[18]:= Suu[j, i] // ToFlavor[red]
         % // EinsteinArray[] // MatrixForm

Out[18]= S^(j i)

Out[19]//MatrixForm=
         ( S^(1 1)  S^(2 1)  S^(3 1) )
         ( S^(1 2)  S^(2 2)  S^(3 2) )
         ( S^(1 3)  S^(2 3)  S^(3 3) )
```

The free indices are always sorted before being passed to `ArrayExpansion`, which arranges the levels in order of the indices it receives.

Using `ArrayExpansion` we could keep the same matrix but switch the order of the indices in the expansion...

```
In[20]:= Suu[i, j] // ToFlavor[red]
         % // ArrayExpansion[red /@ {j, i}] // MatrixForm

Out[20]= S^(i j)

Out[21]//MatrixForm=
         ( S^(1 1)  S^(2 1)  S^(3 1) )
         ( S^(1 2)  S^(2 2)  S^(3 2) )
         ( S^(1 3)  S^(2 3)  S^(3 3) )
```

The following is expanded on a subset of the first sort order index.

```
In[22]:= Suu[i, j] // ToFlavor[red]
         % // EinsteinArray[{1, 2}] // MatrixForm
```

```
Out[22]= S^{i j}
```

```
Out[23]//MatrixForm=
( S^{1 1}   S^{1 2}   S^{1 3} )
( S^{2 1}   S^{2 2}   S^{2 3} )
```

The following is expanded on the same subset of rows and columns.

```
In[24]:= Suu[i, j] // ToFlavor[red]
         % // EinsteinArray[Table[{1, 2}, {2}]] // MatrixForm
```

```
Out[24]= S^{i j}
```

```
Out[25]//MatrixForm=
( S^{1 1}   S^{1 2} )
( S^{2 1}   S^{2 2} )
```

By specifying a list of base index subsets for each index we can obtain a rectangular expansion.

```
In[26]:= Suu[i, j] // ToFlavor[red]
         % // EinsteinArray[{{1, 2}, {1, 2, 3}}] // MatrixForm
```

```
Out[26]= S^{i j}
```

```
Out[27]//MatrixForm=
( S^{1 1}   S^{1 2}   S^{1 3} )
( S^{2 1}   S^{2 2}   S^{2 3} )
```

The following selects two columns instead of two rows.

```
In[28]:= Suu[i, j] // ToFlavor[red]
         % // EinsteinArray[{{1, 2, 3}, {1, 2}}] // MatrixForm
```

```
Out[28]= S^{i j}
```

```
Out[29]//MatrixForm=
( S^{1 1}   S^{1 2} )
( S^{2 1}   S^{2 2} )
( S^{3 1}   S^{3 2} )
```

To obtain the transpose we would switch the indices in the tensor and also switch the order of the base index sets. Remember that the free indices are sorted before being passed to `ArrayExpansion` so `{1, 2}` goes with `i` and `{1, 2, 3}` goes with `j`.

```
In[30]:= Suu[j, i] // ToFlavor[red]
         % // EinsteinArray[{{1, 2}, {1, 2, 3}}] // MatrixForm
```

```
Out[30]= S^{j i}
```

```
Out[31]//MatrixForm=
( S^{1 1}   S^{2 1}   S^{3 1} )
( S^{1 2}   S^{2 2}   S^{3 2} )
```

The following expands a set of equations (or inequalities).

```
In[32]:= yu[i] > Sud[i, j] xu[j] // ToFlavor[rocket]
         % // EinsteinSum[]
         % // EinsteinArray[] // TableForm
```

Out[32]= $y^{i^\dagger} > S^{i^\dagger}{}_{j^\dagger}\, x^{j^\dagger}$

Out[33]= $y^{i^\dagger} > S^{i^\dagger}{}_{1^\dagger}\, x^{1^\dagger} + S^{i^\dagger}{}_{2^\dagger}\, x^{2^\dagger} + S^{i^\dagger}{}_{3^\dagger}\, x^{3^\dagger}$

Out[34]//TableForm=

$$y^{1^\dagger} > S^{1^\dagger}{}_{1^\dagger}\, x^{1^\dagger} + S^{1^\dagger}{}_{2^\dagger}\, x^{2^\dagger} + S^{1^\dagger}{}_{3^\dagger}\, x^{3^\dagger}$$
$$y^{2^\dagger} > S^{2^\dagger}{}_{1^\dagger}\, x^{1^\dagger} + S^{2^\dagger}{}_{2^\dagger}\, x^{2^\dagger} + S^{2^\dagger}{}_{3^\dagger}\, x^{3^\dagger}$$
$$y^{3^\dagger} > S^{3^\dagger}{}_{1^\dagger}\, x^{1^\dagger} + S^{3^\dagger}{}_{2^\dagger}\, x^{2^\dagger} + S^{3^\dagger}{}_{3^\dagger}\, x^{3^\dagger}$$

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

```
In[35]:= DeclareBaseIndices[{1, 2, 3}, {red, {A, B}}]
```

```
In[36]:= Suu[i, red@j]
         % // EinsteinArray[] // MatrixForm
```

Out[36]= $S^{i\,j}$

Out[37]//MatrixForm=

$$\begin{pmatrix} S^{1\,A} & S^{1\,B} \\ S^{2\,A} & S^{2\,B} \\ S^{3\,A} & S^{3\,B} \end{pmatrix}$$

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

```
In[38]:= Suu[i, red@j]
         % // EinsteinArray[{{1, 2}, {B}}] // MatrixForm
```

Out[38]= $S^{i\,j}$

Out[39]//MatrixForm=

$$\begin{pmatrix} S^{1\,B} \\ S^{2\,B} \end{pmatrix}$$

Restore the initial values...

```
In[40]:= ClearTensorShortcuts[{{x, y}, 1}, {S, 2}]
```

```
In[41]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

EinsteinSum

- `EinsteinSum[`*base : Automatic*`][expr]` will perform an Einstein summation on all pairs of matching up/down index pairs. The range of the sum is over the base list, which has the default value of BaseIndices.

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

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

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

See also: `DeclareBaseIndices`, `ArrayExpansion`, `SumExpansion`, `EinsteinArray`, `ToArrayValues`.

Examples

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

Save the settings and declare base indices and flavors.

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

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

EinsteinSum automatically extracts the summation indices.

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

Out[8]= x^i y_i

Out[9]= x^1 y_1 + x^2 y_2 + x^3 y_3
```

EinsteinSum will not sum improper expressions.

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

Out[10]= x^i y^i

Out[11]= x^i y^i
```

EinsteinSum automatically handles flavored indices.

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

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

$$\text{Out[13]=}\quad x^{1}\, y_{1} + x^{2}\, y_{2} + x^{3}\, y_{3}$$

Nonmatching flavors do not sum.

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

$$\text{Out[14]=}\quad x^{i}\, y_{i}$$

$$\text{Out[15]=}\quad x^{i}\, y_{i}$$

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

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

$$\text{Out[16]=}\quad x^{i}\, y_{i}$$

$$\text{Out[17]=}\quad x^{1}\, y_{1} + x^{3}\, y_{3}$$

An error occurs if the base specification is not a subset of the base indices for the index.

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

$$\text{Out[18]=}\quad x^{i}\, y_{i}$$

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

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

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

```
In[20]:= xu[i] xd[i] + Sud[i, j] xu[j] ≠ yu[i] yd[i] + Tud[i, j] yu[j]
         % // EinsteinSum[]
```

$$\text{Out[20]=}\quad S^{i}{}_{j}\, x^{j} + x^{i}\, x_{i} \neq T^{i}{}_{j}\, y^{j} + y^{i}\, y_{i}$$

$$\text{Out[21]=}\quad S^{i}{}_{1}\, x^{1} + S^{i}{}_{2}\, x^{2} + S^{i}{}_{3}\, x^{3} + x^{1}\, x_{1} + x^{2}\, x_{2} + x^{3}\, x_{3} \neq T^{i}{}_{1}\, y^{1} + T^{i}{}_{2}\, y^{2} + T^{i}{}_{3}\, y^{3} + y^{1}\, y_{1} + y^{2}\, y_{2} + y^{3}\, y_{3}$$

It will operate on equations and expressions within arrays.

```
In[22]:= {xu[i] xd[i] + Sud[i, j] xu[j] == yu[i] yd[i] + Tud[i, j] yu[j],
           Sud[i, i] // ToFlavor[red]}
         % // EinsteinSum[]
```

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

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

It sums on dot product expressions and CircleTimes expressions.

```
In[24]:= xu[i].yd[i] // ToFlavor[red]
         % // EinsteinSum[]
```

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

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

```
In[26]:= xu[i]⊗yd[i] // ToFlavor[red]
         % // EinsteinSum[]
```

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

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

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

```
In[28]:= DeclareBaseIndices[{1, 2, 3}, {red, {A, B}}]
```

Now, in the previous example, the two expressions are summed on different base index sets.

```
In[29]:= {xu[i] xd[i] + Sud[i, j] xu[j] == yu[i] yd[i] + Tud[i, j] yu[j],
          Sud[i, i] // ToFlavor[red]}
         % // EinsteinSum[]
```

$$\text{Out[29]=}\quad \left\{S^{i}{}_{j}\,x^{j}+x^{i}\,x_{i}==T^{i}{}_{j}\,y^{j}+y^{i}\,y_{i},\ S^{i}{}_{i}\right\}$$

$$\text{Out[30]=}\quad \{S^{i}{}_{1}\,x^{1}+S^{i}{}_{2}\,x^{2}+S^{i}{}_{3}\,x^{3}+x^{1}\,x_{1}+x^{2}\,x_{2}+x^{3}\,x_{3}==\\ T^{i}{}_{1}\,y^{1}+T^{i}{}_{2}\,y^{2}+T^{i}{}_{3}\,y^{3}+y^{1}\,y_{1}+y^{2}\,y_{2}+y^{3}\,y_{3},\ S^{A}{}_{A}+S^{B}{}_{B}\}$$

```
In[31]:= Suu[a, red@b] Tdd[a, red@b]
         % // EinsteinSum[]
```

$$\text{Out[31]=}\quad S^{a\,b}\,T_{a\,b}$$

$$\text{Out[32]=}\quad S^{1\,A}\,T_{1\,A}+S^{1\,B}\,T_{1\,B}+S^{2\,A}\,T_{2\,A}+S^{2\,B}\,T_{2\,B}+S^{3\,A}\,T_{3\,A}+S^{3\,B}\,T_{3\,B}$$

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

```
In[33]:= Suu[a, red@b] Tdd[a, red@b]
         % // EinsteinSum[{{1, 2}, {B}}]
```

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

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

Restore the initial values...

```
In[35]:= ClearTensorShortcuts[{{x, y}, 1}, {{S, T}, 2}]
```

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

EvaluateDotProducts

- `EvaluateDotProducts[e, g, metricsimplify : True][expr]` expands Dot products of vectors expressed in a given basis e using the metric tensor g. Metric simplification is performed if the default argument metricsimplification is True.

Whether metric simplification is wanted depends upon whether values have been calculated for the metric tensor, or whether they have been calculated for the resulting tensors.

See also: `Dot`, `LinearBreakout`, `BasisDotProductRules`.

Examples

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

Save the settings and declare the base indices and flavors.

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

Define tensor shortcuts.

```
In[7]:= DefineTensorShortcuts[{{e, x, y}, 1}, {g, 2}]
```

Dot products not involving e basis vectors are left unchanged.

```
In[8]:= u.v
        % // EvaluateDotProducts[e, g]

Out[8]= u.v

Out[9]= u.v
```

Or they are partially simplified.

```
In[10]:= u.v /. {u → xu[i] ed[i]}
         % // EvaluateDotProducts[e, g]

Out[10]= (e_i x^i).v

Out[11]= e_i.v x^i
```

If both arguments of the Dot function contain the e basis vectors, then it is fully evaluated.

```
In[12]:= u.v /. {u → xu[i] ed[i], v → yu[j] ed[j]}
         % // EvaluateDotProducts[e, g]

Out[12]= (e_i x^i).(e_j y^j)

Out[13]= x_j y^j
```

Sometimes you may not want metric simplification. Then set the 3rd optional argument to False. For example, suppose you have calculated and stored values for `gdd[i, j]` and `xu[i]` but not `xd[i]`. Then you might prefer the following form.

```
In[14]:= u.v /. {u → xu[i] ed[i], v → yu[j] ed[j]}
         % // EvaluateDotProducts[e, g, False]
```

```
Out[14]= (e_i x^i).(e_j y^j)
```

```
Out[15]= g_i j x^i y^j
```

`EvaluateDotProducts` works on all dot products in an expression.

```
In[16]:= ed[i].ed[j] // ToFlavor[red]
         % // EinsteinArray[] // MatrixForm
         % // EvaluateDotProducts[e, g] // MatrixForm
```

```
Out[16]= e_i.e_j
```

```
Out[17]//MatrixForm=
⎛ e1.e1  e1.e2  e1.e3 ⎞
⎜ e2.e1  e2.e2  e2.e3 ⎟
⎝ e3.e1  e3.e2  e3.e3 ⎠
```

```
Out[18]//MatrixForm=
⎛ g1 1  g1 2  g1 3 ⎞
⎜ g2 1  g2 2  g2 3 ⎟
⎝ g3 1  g3 2  g3 3 ⎠
```

Restore the old values...

```
In[19]:= ClearTensorShortcuts[{{e, x, y}, 1}, {g, 2}]
```

```
In[20]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

EvaluateSlots

- `EvaluateSlots[e, g][expr]` will evaluate all tensor products, involving CircleTimes, on lists of argument slots, which may contain Nulls. e is the label for the basis vectors and g is the label for the metric tensor.

The tensor product and slots will generally contain indicial expressions involving the basis vectors e.

It is assumed that shortcuts have been defined for e and g.

The direct product and slots may contain indicial expressions involving the basis vectors. EvaluateSlots expands and simplifies.

See also: `CircleEvalRule`, `LinearBreakout`, `BasisDotProductRules`, `PushOnto`.

Examples

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

The following evaluates a second order tensor on its two slots.

```
In[3]:= T[x, y]
        % /. {T → Tdd[m, n] eu[m] ⊗ eu[n], x → xu[i] ed[i], y → yu[j] ed[j]}
        % // EvaluateSlots[e, g]
```

```
Out[3]= T[x, y]
```

```
Out[4]= (e^m ⊗ e^n T_mn) [e_i x^i, e_j y^j]
```

```
Out[5]= T_ij x^i y^j
```

If only one slot is filled we obtain a vector.

```
In[6]:= T[, y]
        % /. {T → Tdd[m, n] eu[m] ⊗ eu[n], y → yu[j] ed[j]}
        % // EvaluateSlots[e, g]
        % // EinsteinSum[] // Collect[#, eu[_]] &
```

```
Out[6]= T[Null, y]
```

```
Out[7]= (e^m ⊗ e^n T_mn) [Null, e_j y^j]
```

```
Out[8]= e^m T_mj y^j
```

```
Out[9]= e^1 (T_11 y^1 + T_12 y^2 + T_13 y^3) + e^2 (T_21 y^1 + T_22 y^2 + T_23 y^3) + e^3 (T_31 y^1 + T_32 y^2 + T_33 y^3)
```

You can use it on expressions like the following although CircleEvalRule would work as well. In this case the arguments of EvaluateSlots are not actually used.

```
In[10]:= (u⊗v)[x, y]
         % // EvaluateSlots[e, g]
```

```
Out[10]= (u⊗v)[x, y]
```

```
Out[11]= u.x v.y
```

In the following the direct product and slots are filled with vectors.

```
In[12]:= (u⊗v)[x, y]
         % /. {u → uu[m] ed[m], v → vu[n] ed[n], x → xu[i] ed[i], y → yu[j] ed[j]}
         % // EvaluateSlots[e, g]
         % // EinsteinSum[] // Factor
```

$$\text{Out[12]= } (u \otimes v)[x, y]$$

$$\text{Out[13]= } ((e_m u^m) \otimes (e_n v^n))\left[e_i x^i, e_j y^j\right]$$

$$\text{Out[14]= } u_i v_j x^i y^j$$

$$\text{Out[15]= } (u_1 x^1 + u_2 x^2 + u_3 x^3)(v_1 y^1 + v_2 y^2 + v_3 y^3)$$

```
In[16]:= ClearTensorShortcuts[{{u, v, x, y, e}, 1}, {{T, g}, 2}]
```

ExpandAbsoluteD

- `ExpandAbsoluteD[{x, δ, g, Γ}, {a, b}, permissive : False][expr]` will expand a first order absolute derivative of tensors using coordinates x, Kronecker δ, metric tensor g and Christoffel symbol Γ. The expansion will be done using the dummy indices a and b.
- `ExpandAbsoluteD[{x, δ, g, Γ}, {{a, b}, {c, d}, ...}, permissive : False][expr]` will expand higher order absolute derivatives.

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

`{x, δ, g, Γ}` are the standard set of tensor labels used in all derivative routines. They tell the routines which labels will be considered to represent the coordinate, Kronecker, metric and Christoffel objects. It will often be convenient to define these labels for a notebook and assign them to a short variable, which can then be used in the derivative routines.

A pair of dummy indices is needed for each order of differentiation.

ExpandAbsoluteD is mapped over arrays, equations and sums.

Expressions that contain base indices, Christoffel symbols or partial derivatives will not be expanded.

Expressions that contain Christoffel symbols (from the Γ symbol in the list of tensor symbols) or total derivatives will not be expanded unless the optional parameter, permissive, is set to True. Its default value is False.

Because different dummy indices will be needed in products it is not possible to automatically expand all absolute derivatives in an expression. Sometimes you may have to expand an expression, or you may wish to map to specific parts of expressions.

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

Examples

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

Save the settings.

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

Define the tensor shortcuts and standard tensor labels.

```
In[6]:= DefineTensorShortcuts[{{x, v, S, T}, 1}, {{g, δ, S, T}, 2}, {Γ, 3}]
        labs = {x, δ, g, Γ};
        TensorSymmetry[Γ, 3] = Symmetric[2, 3];
```

Here the absolute derivative of a tensor is first generated in representational form and then expanded. Two dummy indices are needed for each order of differentiation.

```
In[9]:= Su[i]
        AbsoluteD[%, t]
        % // ExpandAbsoluteD[labs, {a, b}]
```

Out[9]= S^i

Out[10]= $\frac{D\,S^i}{d\,t}$

Out[11]= $\frac{dS^i}{dt} + S^a\,\Gamma^i{}_{a\,b}\,\frac{dx^b}{dt}$

For flavored expressions, the intended flavor should also be on the dummy indices.

```
In[12]:= Su[i] // ToFlavor[red]
         AbsoluteD[%, t]
         % // ExpandAbsoluteD[labs, red /@ {a, b}]
```

Out[12]= S^i

Out[13]= $\frac{D\,S^i}{d\,t}$

Out[14]= $\frac{dS^i}{dt} + S^a\,\Gamma^i{}_{a\,b}\,\frac{dx^b}{dt}$

ExpandAbsoluteD maps over arrays, equations and sums...

```
In[15]:= AbsoluteD[#, t] & /@ {a Su[i] - Tu[i], b Su[j] - Tu[j]} == {0, 0} // Thread;
         MapThread[#2 @@ #1 &, {%, {Equal, GreaterEqual}}]
         % // ExpandAbsoluteD[labs, {α, β}]
```

Out[16]= $\left\{a\,\frac{D\,S^i}{d\,t} - \frac{D\,T^i}{d\,t} == 0,\ b\,\frac{D\,S^j}{d\,t} - \frac{D\,T^j}{d\,t} \geq 0\right\}$

Out[17]= $\left\{-\frac{dT^i}{dt} - T^\alpha\,\Gamma^i{}_{\alpha\,\beta}\,\frac{dx^\beta}{dt} + a\left(\frac{dS^i}{dt} + S^\alpha\,\Gamma^i{}_{\alpha\,\beta}\,\frac{dx^\beta}{dt}\right) == 0,\right.$
$\left. -\frac{dT^j}{dt} - T^\alpha\,\Gamma^j{}_{\alpha\,\beta}\,\frac{dx^\beta}{dt} + b\left(\frac{dS^j}{dt} + S^\alpha\,\Gamma^j{}_{\alpha\,\beta}\,\frac{dx^\beta}{dt}\right) \geq 0\right\}$

A correction term is added for each index in the tensor.

```
In[18]:= Sud[i, j] // ToFlavor[red]
         AbsoluteD[%, t]
         % // ExpandAbsoluteD[labs, red /@ {a, b}]
```

Out[18]= $S^i{}_j$

Out[19]= $\frac{D\,S^i{}_j}{d\,t}$

Out[20]= $\frac{dS^i{}_j}{dt} - S^i{}_a\,\Gamma^a{}_{b\,j}\,\frac{dx^b}{dt} + S^a{}_j\,\Gamma^i{}_{a\,b}\,\frac{dx^b}{dt}$

The best way to calculate acceleration is to do it in two steps.

```
In[21]:= AbsoluteD[
          vu[i], t]
         % // ExpandAbsoluteD[labs, {a, b}]
         % /. vu[i_] → TotalD[xu[i], u]
```

$$\text{Out[21]=} \quad \frac{D\, v^i}{d\, t}$$

$$\text{Out[22]=} \quad \frac{d\, v^i}{dt} + v^a\, \Gamma^i{}_{a\, b}\, \frac{dx^b}{dt}$$

$$\text{Out[23]=} \quad \Gamma^i{}_{a\, b}\, \frac{dx^a}{du}\, \frac{dx^b}{dt} + \frac{d^2\, x^i}{du\, dt}$$

To expand a higher order absolute derivative, a pair of dummy indices must be given for each order of derivative.

```
In[24]:= Su[i] // ToFlavor[red]
         AbsoluteD[%, {t, t}]
         % // ExpandAbsoluteD[labs, Map[red, {{a, b}, {c, d}}, {2}]]
         % // SimplifyTensorSum
```

$$\text{Out[24]=} \quad S^i$$

$$\text{Out[25]=} \quad \frac{D^2\, S^i}{d\, t\, d\, t}$$

$$\text{Out[26]=} \quad \frac{d^2\, S^i}{dt\, dt} + \Gamma^i{}_{a\, b}\, \frac{dS^a}{dt}\, \frac{dx^b}{dt} + \Gamma^i{}_{c\, d} \left(\frac{dS^c}{dt} + S^a\, \Gamma^c{}_{a\, b}\, \frac{dx^b}{dt} \right) \frac{dx^d}{dt} + S^a \left(\Gamma^i{}_{a\, b}\, \frac{d^2\, x^b}{dt\, dt} + \frac{dx^b}{dt}\, \frac{d\Gamma^i{}_{a\, b}}{dt} \right)$$

$$\text{Out[27]=} \quad \frac{d^2\, S^i}{dt\, dt} + 2\, \Gamma^i{}_{a\, b}\, \frac{dS^a}{dt}\, \frac{dx^b}{dt} + S^a\, \Gamma^i{}_{a\, b}\, \frac{d^2\, x^b}{dt\, dt} + S^a\, \Gamma^d{}_{a\, b}\, \Gamma^i{}_{d\, c}\, \frac{dx^b}{dt}\, \frac{dx^c}{dt} + S^a\, \frac{dx^b}{dt}\, \frac{d\Gamma^i{}_{a\, b}}{dt}$$

```
In[28]:= Su[i] + Tu[i] // ToFlavor[red]
         AbsoluteD[%, {t, t}]
         % // ExpandAbsoluteD[labs, Map[red, {{a, b}, {c, d}}, {2}]]
         % // TensorSimplify // FullSimplify
```

$$\text{Out[28]=} \quad S^i + T^i$$

$$\text{Out[29]=} \quad \frac{D^2\, S^i}{d\, t\, d\, t} + \frac{D^2\, T^i}{d\, t\, d\, t}$$

$$\text{Out[30]=} \quad \frac{d^2\, S^i}{dt\, dt} + \frac{d^2\, T^i}{dt\, dt} + \Gamma^i{}_{a\, b}\, \frac{dS^a}{dt}\, \frac{dx^b}{dt} + \Gamma^i{}_{a\, b}\, \frac{dT^a}{dt}\, \frac{dx^b}{dt} + \Gamma^i{}_{c\, d} \left(\frac{dS^c}{dt} + S^a\, \Gamma^c{}_{a\, b}\, \frac{dx^b}{dt} \right) \frac{dx^d}{dt} + \Gamma^i{}_{c\, d} \left(\frac{dT^c}{dt} + T^a\, \Gamma^c{}_{a\, b}\, \frac{dx^b}{dt} \right) \frac{dx^d}{dt} + S^a \left(\Gamma^i{}_{a\, b}\, \frac{d^2\, x^b}{dt\, dt} + \frac{dx^b}{dt}\, \frac{d\Gamma^i{}_{a\, b}}{dt} \right) + T^a \left(\Gamma^i{}_{a\, b}\, \frac{d^2\, x^b}{dt\, dt} + \frac{dx^b}{dt}\, \frac{d\Gamma^i{}_{a\, b}}{dt} \right)$$

$$\text{Out[31]=} \quad \frac{d^2\, S^i}{dt\, dt} + \frac{d^2\, T^i}{dt\, dt} + \Gamma^i{}_{a\, b} \left(2 \left(\frac{dS^a}{dt} + \frac{dT^a}{dt} \right) \frac{dx^b}{dt} + (S^a + T^a)\, \frac{d^2\, x^b}{dt\, dt} \right) + (S^a + T^a)\, \frac{dx^b}{dt} \left(\Gamma^c{}_{a\, b}\, \Gamma^i{}_{c\, d}\, \frac{dx^d}{dt} + \frac{d\Gamma^i{}_{a\, b}}{dt} \right)$$

The same must be done for products of absolute derivatives

```
In[32]:= AbsoluteD[Su[i], t] AbsoluteD[Tu[j], t]
         % // ExpandAbsoluteD[labs, {{a, b}, {c, d}}]
```

$$\text{Out[32]=} \quad \frac{D\,S^i}{d\,t}\,\frac{D\,T^j}{d\,t}$$

$$\text{Out[33]=} \quad \left(\frac{dS^i}{dt} + S^a\,\Gamma^i{}_{a\,b}\,\frac{dx^b}{dt}\right)\left(\frac{dT^j}{dt} + T^c\,\Gamma^j{}_{c\,d}\,\frac{dx^d}{dt}\right)$$

This doesn't match because there is a wrong number of indices.

```
In[34]:= AbsoluteD[Su[i], t] AbsoluteD[Tu[j], t] == 0
         % // ExpandAbsoluteD[labs, {a, b}]
```

$$\text{Out[34]=} \quad \frac{D\,S^i}{d\,t}\,\frac{D\,T^j}{d\,t} == 0$$

$$\text{Out[35]=} \quad \frac{D\,S^i}{d\,t}\,\frac{D\,T^j}{d\,t} == 0$$

The following matches only one of the terms.

```
In[36]:= AbsoluteD[Suu[i, j], t] + AbsoluteD[Su[i], t] AbsoluteD[Tu[j], t] == 0
         % // ExpandAbsoluteD[labs, {a, b}]
```

$$\text{Out[36]=} \quad \frac{D\,S^{i\,j}}{d\,t} + \frac{D\,S^i}{d\,t}\,\frac{D\,T^j}{d\,t} == 0$$

$$\text{Out[37]=} \quad \frac{D\,S^i}{d\,t}\,\frac{D\,T^j}{d\,t} + \frac{dS^{i\,j}}{dt} + S^{a\,j}\,\Gamma^i{}_{a\,b}\,\frac{dx^b}{dt} + S^{i\,a}\,\Gamma^j{}_{a\,b}\,\frac{dx^b}{dt} == 0$$

```
In[38]:= AbsoluteD[Su[i] + Tu[i], t] AbsoluteD[Su[j] - Tu[j], t]
         % // ExpandAbsoluteD[labs, {{a, b}, {c, d}}]
```

$$\text{Out[38]=} \quad \left(\frac{D\,S^i}{d\,t} + \frac{D\,T^i}{d\,t}\right)\left(\frac{D\,S^j}{d\,t} - \frac{D\,T^j}{d\,t}\right)$$

$$\text{Out[39]=} \quad \left(\frac{dS^j}{dt} + S^a\,\Gamma^j{}_{a\,b}\,\frac{dx^b}{dt}\right)\left(\frac{dT^i}{dt} + T^c\,\Gamma^i{}_{c\,d}\,\frac{dx^d}{dt}\right) + \left(\frac{dS^i}{dt} + S^a\,\Gamma^i{}_{a\,b}\,\frac{dx^b}{dt}\right)\left(\frac{dS^j}{dt} + S^c\,\Gamma^j{}_{c\,d}\,\frac{dx^d}{dt}\right) -$$
$$\left(\frac{dS^i}{dt} + S^a\,\Gamma^i{}_{a\,b}\,\frac{dx^b}{dt}\right)\left(\frac{dT^j}{dt} + T^c\,\Gamma^j{}_{c\,d}\,\frac{dx^d}{dt}\right) - \left(\frac{dT^i}{dt} + T^a\,\Gamma^i{}_{a\,b}\,\frac{dx^b}{dt}\right)\left(\frac{dT^j}{dt} + T^c\,\Gamma^j{}_{c\,d}\,\frac{dx^d}{dt}\right)$$

Here is the second derivative of a tensor product.

```
In[40]:= Sd[i] Tu[j]
         AbsoluteD[%, {t, t}]
         % // ExpandAbsoluteD[labs, {{a, b}, {c, d}}];
         % // TensorSimplify // Simplify
```

$$\text{Out[40]=}\quad S_i\, T^j$$

$$\text{Out[41]=}\quad 2\,\frac{D\,S_i}{d\,t}\,\frac{D\,T^j}{d\,t}+\frac{D^2\,T^j}{d\,t\,d\,t}\,S_i+\frac{D^2\,S_i}{d\,t\,d\,t}\,T^j$$

$$\begin{aligned}\text{Out[43]=}\quad & S_i\,\frac{d^2\,T^j}{dt\,dt}+2\,S_i\,\Gamma^j{}_{ab}\,\frac{dT^a}{dt}\,\frac{dx^b}{dt}-2\,S_a\,\Gamma^a{}_{bi}\,\frac{dT^j}{dt}\,\frac{dx^b}{dt}+S_i\,T^a\,\Gamma^j{}_{ab}\,\frac{d^2\,x^b}{dt\,dt}-\\ & 2\,S_a\,T^c\,\Gamma^a{}_{bi}\,\Gamma^j{}_{cd}\,\frac{dx^b}{dt}\,\frac{dx^d}{dt}+S_i\,T^a\,\Gamma^c{}_{ab}\,\Gamma^j{}_{cd}\,\frac{dx^b}{dt}\,\frac{dx^d}{dt}+2\,\frac{dS_i}{dt}\left(\frac{dT^j}{dt}+T^c\,\Gamma^j{}_{cd}\,\frac{dx^d}{dt}\right)+\\ & T^j\left(\frac{d^2\,S_i}{dt\,dt}-\Gamma^a{}_{bi}\left(2\,\frac{dS_a}{dt}\,\frac{dx^b}{dt}+S_a\,\frac{d^2\,x^b}{dt\,dt}\right)+S_a\,\frac{dx^b}{dt}\left(\Gamma^a{}_{bc}\,\Gamma^c{}_{di}\,\frac{dx^d}{dt}-\frac{d\Gamma^a{}_{bi}}{dt}\right)\right)+\\ & S_i\,T^a\,\frac{dx^b}{dt}\,\frac{d\Gamma^j{}_{ab}}{dt}\end{aligned}$$

Calculation of velocity and acceleration tensors in spherical coordinates: ρ is the distance from the origin to a point, θ is the angle from the North pole, and ϕ is the rotation about the z axis.

1) Enter the metric matrix for spherical coordinates in symbolic coordinate form.

2) Convert to coordinate positions with label x (the default).

3) Set the metric tensor values. The inverse is also calculated and set.

4) Calculate and set the Christoffel symbols using the label Γ to represent Christoffel symbols.

```
In[44]:= cmetric = DiagonalMatrix[{1, ρ^2, ρ^2 Sin[θ]^2}];
         (metric = cmetric // CoordinatesToTensors[{ρ, θ, ϕ}]) // MatrixForm
         MapThread[SetTensorValues[#1, #2] &,
           {{gdd[a, b], guu[a, b]}, {metric, Inverse@metric}}];
         MapThread[SetTensorValues[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b,]}, CalculateChristoffels[labs]}];
```

$$\text{Out[45]//MatrixForm=}\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & (x^1)^2 & 0 \\ 0 & 0 & \text{Sin}[x^2]^2\,(x^1)^2 \end{pmatrix}$$

The velocity is the derivative of the coordinate positions.

```
In[48]:= TotalD[xu[i], t]
```

$$\text{Out[48]=}\quad \frac{dx^i}{dt}$$

We set tensor values for the velocity and we set coordinate position rules for an object rotating around equatorial plane at a radius of R and an angular velocity ω.

```
In[49]:= SetTensorValues[vu[i], TotalD[xu[i], t] // ToArrayValues[]]
         SetTensorValueRules[xu[i], {R, π / 2, ω t}]
```

Now it is trivial to calculate the acceleration by taking the absolute derivative of the velocity.

1) Take the absolute derivative of the velocity.

2) Expand it in terms of Christoffel symbols and total derivatives.

3) Expand to the components in terms of the coordinate vectors.

4) Substitute coordinate values.

```
In[51]:= AbsoluteD[vu[m], t]
         % // ExpandAbsoluteD[labs, {μ, ν}]
         (% // EinsteinSum[] // EinsteinArray[]) /. TensorValueRules[Γ]
         % // UseCoordinates[{r, θ, ϕ}]
```

$$\text{Out[51]=} \quad \frac{D\,v^m}{d\,t}$$

$$\text{Out[52]=} \quad \frac{d\,v^m}{dt} + v^\mu\,\Gamma^m{}_{\mu\nu}\,\frac{d\,x^\nu}{dt}$$

$$\text{Out[53]=} \quad \left\{\frac{d^2\,x^1}{dt\,dt} - x^1\left(\frac{d\,x^2}{dt}\right)^2 - \text{Sin}[x^2]^2\,x^1\left(\frac{d\,x^3}{dt}\right)^2,\right.$$
$$\left.\frac{2\,\frac{dx^1}{dt}\,\frac{dx^2}{dt}}{x^1} + \frac{d^2\,x^2}{dt\,dt} - \text{Cos}[x^2]\,\text{Sin}[x^2]\left(\frac{d\,x^3}{dt}\right)^2,\ \frac{2\,\frac{dx^1}{dt}\,\frac{dx^3}{dt}}{x^1} + 2\,\text{Cot}[x^2]\,\frac{d\,x^2}{dt}\,\frac{d\,x^3}{dt} + \frac{d^2\,x^3}{dt\,dt}\right\}$$

$$\text{Out[54]=} \quad \left\{\frac{d^2\,r}{dt^2} - r\left(\frac{d\theta}{dt}\right)^2 - r\left(\frac{d\phi}{dt}\right)^2\text{Sin}[\theta]^2,\right.$$
$$\left.\frac{2\,\frac{dr}{dt}\,\frac{d\theta}{dt}}{r} + \frac{d^2\theta}{dt^2} - \text{Cos}[\theta]\left(\frac{d\phi}{dt}\right)^2\text{Sin}[\theta],\ \frac{2\,\frac{dr}{dt}\,\frac{d\phi}{dt}}{r} + 2\,\text{Cot}[\theta]\,\frac{d\theta}{dt}\,\frac{d\phi}{dt} + \frac{d^2\phi}{dt^2}\right\}$$

Which is actually the result we can find in any vector calculus books. For the orbiting object we specified above, we can just use `ToArrayValues`, which expands and substitutes all rules. We obtain a general form because *Mathematica* does not know that ω and R are constant.

```
In[55]:= AbsoluteD[vu[i], t]
         % // ExpandAbsoluteD[labs, {μ, ν}]
         % // ToArrayValues[]
```

$$\text{Out[55]=} \quad \frac{D\,v^i}{d\,t}$$

$$\text{Out[56]=} \quad \frac{d\,v^i}{dt} + v^\mu\,\Gamma^i{}_{\mu\nu}\,\frac{d\,x^\nu}{dt}$$

$$\text{Out[57]=} \quad \left\{\frac{d^2\,R}{dt^2} - R\left(\omega + t\,\frac{d\omega}{dt}\right)^2,\ 0,\ 2\,\frac{d\omega}{dt} + \frac{2\,\frac{dR}{dt}\,\left(\omega + t\,\frac{d\omega}{dt}\right)}{R} + t\,\frac{d^2\,\omega}{dt^2}\right\}$$

If we wish ω and R to be regarded as constants...

```
In[58]:= SetAttributes[{R, ω}, Constant];
         AbsoluteD[vu[m], t]
         % // ExpandAbsoluteD[labs, {μ, ν}]
         % // ToArrayValues[]
         ClearAll[R, ω]
```

$$\text{Out[59]=} \quad \frac{D\,v^m}{d\,t}$$

$$\text{Out[60]=} \quad \frac{d\,v^m}{dt} + v^\mu\,\Gamma^m{}_{\mu\nu}\,\frac{d\,x^\nu}{dt}$$

$$\text{Out[61]=} \quad \{-R\,\omega^2,\ 0,\ 0\}$$

Higher order derivatives are best done in one step.

```
In[63]:= AbsoluteD[Td[a], {s, t}]
         step1 = % // ExpandAbsoluteD[labs, {{b, c}, {d, e}}]
```

$$\text{Out[63]=} \quad \frac{D^2\, T_a}{d\,s\,d\,t}$$

$$\text{Out[64]=} \quad \frac{d^2\, T_a}{ds\, dt} - \Gamma^b{}_{ca}\, \frac{dT_b}{dt}\, \frac{dx^c}{ds} - \Gamma^d{}_{ea} \left(\frac{dT_d}{ds} - T_b\, \Gamma^b{}_{cd}\, \frac{dx^c}{ds} \right) \frac{dx^e}{dt} - T_b \left(\Gamma^b{}_{ca}\, \frac{d^2 x^c}{ds\, dt} + \frac{dx^c}{ds}\, \frac{d\Gamma^b{}_{ca}}{dt} \right)$$

The following is the same calculation done in two steps. The second expansion must be permissive because the input contains a Christoffel symbo and total derivativesl.

```
In[65]:= AbsoluteD[Td[a], s]
         % // ExpandAbsoluteD[labs, {b, c}]
         AbsoluteD[%, t]
         step2 = % // ExpandAbsoluteD[labs, {d, e}, True]
```

$$\text{Out[65]=} \quad \frac{D\, T_a}{d\,s}$$

$$\text{Out[66]=} \quad \frac{dT_a}{ds} - T_b\, \Gamma^b{}_{ca}\, \frac{dx^c}{ds}$$

$$\text{Out[67]=} \quad \frac{D\, \frac{dT_a}{ds}}{d\,t} - \frac{D\, T_b}{d\,t}\, \Gamma^b{}_{ca}\, \frac{dx^c}{ds} - T_b \left(\frac{D\, \frac{dx^c}{ds}}{d\,t}\, \Gamma^b{}_{ca} + \frac{D\, \Gamma^b{}_{ca}}{d\,t}\, \frac{dx^c}{ds} \right)$$

$$\text{Out[68]=} \quad \frac{d^2\, T_a}{ds\, dt} - \Gamma^d{}_{ea}\, \frac{dT_d}{ds}\, \frac{dx^e}{dt} - \Gamma^b{}_{ca}\, \frac{dx^c}{ds} \left(\frac{dT_b}{dt} - T_d\, \Gamma^d{}_{eb}\, \frac{dx^e}{dt} \right) - T_b\, \Gamma^b{}_{ca} \left(\frac{d^2 x^c}{ds\, dt} + \Gamma^c{}_{de}\, \frac{dx^d}{ds}\, \frac{dx^e}{dt} \right) - T_b\, \frac{dx^c}{ds} \left(\Gamma^b{}_{de}\, \Gamma^d{}_{ca}\, \frac{dx^e}{dt} - \Gamma^b{}_{cd}\, \Gamma^d{}_{ea}\, \frac{dx^e}{dt} - \Gamma^b{}_{da}\, \Gamma^d{}_{ec}\, \frac{dx^e}{dt} + \frac{d\Gamma^b{}_{ca}}{dt} \right)$$

We can check that they are the same.

```
In[69]:= step2 - step1
         Nest[TensorSimplify, %, 2]
```

$$\begin{aligned}\text{Out[69]=} \quad & \Gamma^b{}_{ca}\, \frac{dT_b}{dt}\, \frac{dx^c}{ds} - \Gamma^d{}_{ea}\, \frac{dT_d}{ds}\, \frac{dx^e}{dt} + \Gamma^d{}_{ea} \left(\frac{dT_d}{ds} - T_b\, \Gamma^b{}_{cd}\, \frac{dx^c}{ds} \right) \frac{dx^e}{dt} - \\ & \Gamma^b{}_{ca}\, \frac{dx^c}{ds} \left(\frac{dT_b}{dt} - T_d\, \Gamma^d{}_{eb}\, \frac{dx^e}{dt} \right) - T_b\, \Gamma^b{}_{ca} \left(\frac{d^2 x^c}{ds\, dt} + \Gamma^c{}_{de}\, \frac{dx^d}{ds}\, \frac{dx^e}{dt} \right) - \\ & T_b\, \frac{dx^c}{ds} \left(\Gamma^b{}_{de}\, \Gamma^d{}_{ca}\, \frac{dx^e}{dt} - \Gamma^b{}_{cd}\, \Gamma^d{}_{ea}\, \frac{dx^e}{dt} - \Gamma^b{}_{da}\, \Gamma^d{}_{ec}\, \frac{dx^e}{dt} + \frac{d\Gamma^b{}_{ca}}{dt} \right) + \\ & T_b \left(\Gamma^b{}_{ca}\, \frac{d^2 x^c}{ds\, dt} + \frac{dx^c}{ds}\, \frac{d\Gamma^b{}_{ca}}{dt} \right)\end{aligned}$$

$$\text{Out[70]=} \quad 0$$

The following expressions will not be expanded because the first contains a base index and the second contains a partial derivative.

```
In[71]:= Td[1]
         AbsoluteD[%, t]
         % // ExpandAbsoluteD[labs, {a, b}]
```

Out[71]= T_1

Out[72]= $\frac{D\,T_1}{d\,t}$

```
ExpandAbsoluteD::nottensor :
 An absolute derivative , D T1/dt, cannot be expanded because
   Tensorial cannot assess the tensor nature of the expression.
```

```
Out[73]= $Aborted
```

```
In[74]:= PartialD[labs][Td[a], xu[i]]
         AbsoluteD[%, t]
         % // ExpandAbsoluteD[labs, {c, d}]
```

Out[74]= $\frac{\partial T_a}{\partial x^i}$

Out[75]= $\frac{D\,\frac{\partial T_a}{\partial x^i}}{d\,t}$

```
ExpandAbsoluteD::nottensor :
 An absolute derivative , D ∂Ta/∂xi/dt, cannot be expanded because
   Tensorial cannot assess the tensor nature of the expression.
```

```
Out[76]= $Aborted
```

```
In[77]:= ClearTensorValues[
           {gdd[a, v], guu[a, b], vu[i], xu[i], Γudd[i, j, k], Γddd[i, j, k]}];
         ClearTensorShortcuts[{{x, v, S, T}, 1}, {{g, δ, S, T}, 2}, {Γ, 3}]
```

```
In[79]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors, metric, step1, step2]
```

ExpandCovariantD

- `ExpandCovariantD[{x, δ, g, Γ}, a, permissive : False][expr]` will expand first order covariant derivatives of tensors using x as the label for the coordinate positions, δ as the label for the Kronecker, g as the label for the metric tensor and Γ as the lavel for Christoffel symbols. The introduced dummy index will be a.
- `ExpandCovariantD[{x, δ, g, Γ}, {a, b, ...}, permissive : False][expr]` expands higher order covariant derivatives using the list of dummy indices.

ExpandCovariantD is mapped over arrays, equations and Plus.

ExpandCovariantD is intended to work only on tensor expressions. It will not expand expressions containing base indices or unexpanded partial derivatives. It will not expand expressions containing expanded partial derivatives or Christoffel symbols (from Γ in the list of labels) unless the user sets the optional parameter permissive to True. This might be done if the expression contain results from previous covariant derivatives.

The expansion is done for a 'coordinate basis' or holonomic system. in which case the Christoffel up symbols, with the first index up, are the same as the 'connection coefficients'.

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

See also: `CovariantD`, `SetChristoffelValueRules`, `PartialD`, `AbsoluteD`, `TotalD`, `Tensor`.

Examples

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

Save the old settings.

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

Define standard labels and shortcuts.

```
In[6]:= labs = {x, δ, g, Γ};
        DefineTensorShortcuts[{{x, y, S, T}, 1}, {{g, δ, S, T}, 2}, {{Γ, Λ, zero}, 3}]
        DeclareZeroTensor[zero]
        TensorLabelFormat[zero, 0]
```

Here is a covariant derivative of a second order tensor and its expansion in terms of a partial derivatives and Christoffel symbols using the dummy index a. There is a Christoffel correction term for each index.

```
In[10]:= Tuu[i, j]
         CovariantD[%, k]
         % // ExpandCovariantD[labs, a]
```

Out[10]= $T^{i\,j}$

Out[11]= $T^{i\,j}{}_{;k}$

Out[12]= $T^{a\,j}\,\Gamma^{i}{}_{k\,a} + T^{i\,a}\,\Gamma^{j}{}_{k\,a} + \frac{\partial T^{i\,j}}{\partial x^{k}}$

With flavored expressions the intended flavor must also be on the covariant dummy index.

```
In[13]:= Tud[i, j] // ToFlavor[red]
         CovariantD[%, red@k]
         % // ExpandCovariantD[labs, red@a]
```

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

Out[14]= $T^{i}{}_{j;k}$

Out[15]= $-T^{i}{}_{a}\,\Gamma^{a}{}_{k\,j} + T^{a}{}_{j}\,\Gamma^{i}{}_{k\,a} + \frac{\partial T^{i}{}_{j}}{\partial x^{k}}$

A dummy index name must be supplied for each covariant differentiation.

```
In[16]:= Tuu[i, j] // ToFlavor[red]
         CovariantD[%, red /@ {m, n}]
         % // ExpandCovariantD[labs, red /@ {a, b}]
```

Out[16]= $T^{i\,j}$

Out[17]= $T^{i\,j}{}_{;m\,n}$

Out[18]= $\Gamma^{i}{}_{m\,a}\,\frac{\partial T^{a\,j}}{\partial x^{n}} + \Gamma^{i}{}_{n\,b}\left(T^{a\,j}\,\Gamma^{b}{}_{m\,a} + T^{b\,a}\,\Gamma^{j}{}_{m\,a} + \frac{\partial T^{b\,j}}{\partial x^{m}}\right) +$
$\Gamma^{j}{}_{m\,a}\,\frac{\partial T^{i\,a}}{\partial x^{n}} + \Gamma^{j}{}_{n\,b}\left(T^{i\,a}\,\Gamma^{b}{}_{m\,a} + T^{a\,b}\,\Gamma^{i}{}_{m\,a} + \frac{\partial T^{i\,b}}{\partial x^{m}}\right) + \frac{\partial^{2} T^{i\,j}}{\partial x^{n}\,\partial x^{m}} -$
$\Gamma^{b}{}_{n\,m}\left(T^{a\,j}\,\Gamma^{i}{}_{b\,a} + T^{i\,a}\,\Gamma^{j}{}_{b\,a} + \frac{\partial T^{i\,j}}{\partial x^{b}}\right) + T^{a\,j}\,\frac{\partial \Gamma^{i}{}_{m\,a}}{\partial x^{n}} + T^{i\,a}\,\frac{\partial \Gamma^{j}{}_{m\,a}}{\partial x^{n}}$

We could use different symbols for the coordinate and Christoffel symbols. This uses Λ for the connection and y for the coordinates.

```
In[19]:= Tdd[i, j] // ToFlavor[red]
         CovariantD[%, red@k]
         % // ExpandCovariantD[{y, δ, g, Λ}, red@a]
```

Out[19]= $T_{i\,j}$

Out[20]= $T_{i\,j;k}$

Out[21]= $-T_{a\,j}\,\Lambda^{a}{}_{k\,i} - T_{i\,a}\,\Lambda^{a}{}_{k\,j} + \frac{\partial T_{i\,j}}{\partial y^{k}}$

If a tensor has no covariant derivative indices then `ExpandCovariantD` does nothing. In the following, only one term is expanded.

```
In[22]:= 2 Tud[i, j] + CovariantD[Tu[i], j]
         % // ExpandCovariantD[labs, a]
```

$$\text{Out[22]=}\quad T^{i}{}_{;j} + 2\,T^{i}{}_{j}$$

$$\text{Out[23]=}\quad 2\,T^{i}{}_{j} + T^{a}\,\Gamma^{i}{}_{j\,a} + \frac{\partial T^{i}}{\partial x^{j}}$$

The following is an example of a calculation with covariant derivatives. We set the metric for a spherical coordinate system and express it in terms of coordinate positions.

```
In[24]:= metric = DiagonalMatrix[{1, r^2, r^2 Sin[θ]^2}];
         (tmetric = metric // CoordinatesToTensors[{r, θ, ϕ}]) // MatrixForm
         MapThread[SetTensorValueRules[#1, #2] &,
           {{gdd[a, b], guu[a, b]}, {tmetric, Inverse@tmetric}}];
```

Out[25]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & (x^1)^2 & 0 \\ 0 & 0 & \text{Sin}[x^2]^2\,(x^1)^2 \end{pmatrix}$$

We then calculate and set the Christoffel values...

```
In[27]:= MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]}, CalculateChristoffels[labs]}];
```

The following displays the independent nonzero values of the up Christoffel symbols.

```
In[28]:= SelectedTensorRules[Γ, Γudd[_, a_, b_] /; OrderedQ[{a, b}]] // TableForm
```

Out[28]//TableForm=

$$\Gamma^{1}{}_{2\,2} \to -x^{1}$$

$$\Gamma^{1}{}_{3\,3} \to -\text{Sin}[x^2]^2\,x^{1}$$

$$\Gamma^{2}{}_{1\,2} \to (x^{1})^{-1}$$

$$\Gamma^{2}{}_{3\,3} \to -\text{Cos}[x^2]\,\text{Sin}[x^2]$$

$$\Gamma^{3}{}_{1\,3} \to (x^{1})^{-1}$$

$$\Gamma^{3}{}_{2\,3} \to \text{Cot}[x^2]$$

We then

1) Set the coordinate position values to coordinate symbols via rules.

2) Take the covariant derivative.

3) Expand the covariant derivative in terms of Christoffel symbols and the coordinate positions.

4) Expand the expression into an array making all substitutions.

5) Clear the rules for the coordinate positions.

```
In[29]:= SetTensorValueRules[xu[i], {r, θ, ϕ}];
         CovariantD[Tu[i], k]
         % // ExpandCovariantD[labs, a]
         % // ToArrayValues[]
         ClearTensorValues[xu[i]]
```

$$\text{Out[30]= } T^{i}{}_{;k}$$

$$\text{Out[31]= } T^{a}\,\Gamma^{i}{}_{k\,a} + \frac{\partial T^{i}}{\partial x^{k}}$$

$$\text{Out[32]= } \left\{\left\{\frac{\partial T^{1}}{\partial r}, -r\,T^{2} + \frac{\partial T^{1}}{\partial \theta}, -r\,\text{Sin}[\theta]^{2}\,T^{3} + \frac{\partial T^{1}}{\partial \phi}\right\},\right.$$
$$\left\{\frac{T^{2}}{r} + \frac{\partial T^{2}}{\partial r}, \frac{T^{1}}{r} + \frac{\partial T^{2}}{\partial \theta}, -\text{Cos}[\theta]\,\text{Sin}[\theta]\,T^{3} + \frac{\partial T^{2}}{\partial \phi}\right\},$$
$$\left.\left\{\frac{T^{3}}{r} + \frac{\partial T^{3}}{\partial r}, \text{Cot}[\theta]\,T^{3} + \frac{\partial T^{3}}{\partial \theta}, \frac{T^{1}}{r} + \text{Cot}[\theta]\,T^{2} + \frac{\partial T^{3}}{\partial \phi}\right\}\right\}$$

That gives us the covariant derivative components of T in terms of the contravariant T components.

A tensor times a covariant derivative.

```
In[34]:= Tdu[k, j] CovariantD[Sd[k], m]
         % // ExpandCovariantD[labs, a]
```

$$\text{Out[34]= } S_{k;m}\,T_{k}{}^{j}$$

$$\text{Out[35]= } T_{k}{}^{j}\left(-S_{a}\,\Gamma^{a}{}_{m\,k} + \frac{\partial S_{k}}{\partial x^{m}}\right)$$

The covariant derivative of a product of tensors and a constant.

```
In[36]:= 2 π q Sd[k] Tdu[j, k]
         CovariantD[%, m]
         % // ExpandCovariantD[labs, a]
```

$$\text{Out[36]= } 2\,\pi\,q\,S_{k}\,T_{j}{}^{k}$$

$$\text{Out[37]= } 2\,\pi\,q\left(T_{j}{}^{k}{}_{;m}\,S_{k} + S_{k;m}\,T_{j}{}^{k}\right)$$

$$\text{Out[38]= } 2\,\pi\,q\,T_{j}{}^{k}\left(-S_{a}\,\Gamma^{a}{}_{m\,k} + \frac{\partial S_{k}}{\partial x^{m}}\right) + 2\,\pi\,q\,S_{k}\left(-T_{a}{}^{k}\,\Gamma^{a}{}_{m\,j} + T_{j}{}^{a}\,\Gamma^{k}{}_{m\,a} + \frac{\partial T_{j}{}^{k}}{\partial x^{m}}\right)$$

```
In[39]:= Sd[a] Tu[b]
         CovariantD[%, {c, d}]
         % // ExpandCovariantD[labs, {e, f}]
```

$$\text{Out[39]= } S_{a}\,T^{b}$$

$$\text{Out[40]= } S_{a;d}\,T^{b}{}_{;c} + S_{a;c}\,T^{b}{}_{;d} + T^{b}{}_{;c\,d}\,S_{a} + S_{a;c\,d}\,T^{b}$$

$$\text{Out[41]= } \left(-S_{e}\,\Gamma^{e}{}_{d\,a} + \frac{\partial S_{a}}{\partial x^{d}}\right)\left(T^{f}\,\Gamma^{b}{}_{c\,f} + \frac{\partial T^{b}}{\partial x^{c}}\right) + \left(-S_{e}\,\Gamma^{e}{}_{c\,a} + \frac{\partial S_{a}}{\partial x^{c}}\right)\left(T^{f}\,\Gamma^{b}{}_{d\,f} + \frac{\partial T^{b}}{\partial x^{d}}\right) +$$
$$S_{a}\left(\frac{\partial^{2} T^{b}}{\partial x^{d}\,\partial x^{c}} - \Gamma^{f}{}_{d\,c}\left(T^{e}\,\Gamma^{b}{}_{f\,e} + \frac{\partial T^{b}}{\partial x^{f}}\right) + \Gamma^{b}{}_{c\,e}\,\frac{\partial T^{e}}{\partial x^{d}} + \Gamma^{b}{}_{d\,f}\left(T^{e}\,\Gamma^{f}{}_{c\,e} + \frac{\partial T^{f}}{\partial x^{c}}\right) + T^{e}\,\frac{\partial \Gamma^{b}{}_{c\,e}}{\partial x^{d}}\right) +$$
$$T^{b}\left(\frac{\partial^{2} S_{a}}{\partial x^{d}\,\partial x^{c}} - \Gamma^{f}{}_{d\,c}\left(-S_{e}\,\Gamma^{e}{}_{f\,a} + \frac{\partial S_{a}}{\partial x^{f}}\right) - \Gamma^{e}{}_{c\,a}\,\frac{\partial S_{e}}{\partial x^{d}} - \Gamma^{f}{}_{d\,a}\left(-S_{e}\,\Gamma^{e}{}_{c\,f} + \frac{\partial S_{f}}{\partial x^{c}}\right) - S_{e}\,\frac{\partial \Gamma^{e}{}_{c\,a}}{\partial x^{d}}\right)$$

Now, let's try the well known identity about the covariant derivative of the metric tensor, using the metric tensor and Christoffel symbols from above. We equate the covariant derivative to the 3rd order zero tensor.

```
In[42]:= SetTensorValueRules[δud[i, j], IdentityMatrix[NDim]]
         gdd[m, n]
         CovariantD[%, i] == zeroddd[n, m, i]
         step1 = % // ExpandCovariantD[labs, a]
         ToArrayValues[] /@ % // MatrixForm
```

$$\text{Out[43]=}\quad g_{mn}$$

$$\text{Out[44]=}\quad g_{mn;i} == 0_{nmi}$$

$$\text{Out[45]=}\quad -g_{an}\,\Gamma^{a}{}_{im} - g_{ma}\,\Gamma^{a}{}_{in} + \frac{\partial g_{mn}}{\partial x^{i}} == 0_{nmi}$$

Out[46]//MatrixForm=

True

The double covariant derivative of a scalar still requires two expansion indices, but the first one is not used and can even be a Null.

```
In[47]:= Tensor[ϕ]
         CovariantD[%, {a, b}]
         % // ExpandCovariantD[labs, {, d}]
```

$$\text{Out[47]=}\quad \phi$$

$$\text{Out[48]=}\quad \phi_{;ab}$$

$$\text{Out[49]=}\quad \frac{\partial^{2}\phi}{\partial x^{b}\,\partial x^{a}} - \Gamma^{d}{}_{ba}\,\frac{\partial \phi}{\partial x^{d}}$$

That usage allows us to use the same ExpandCovariantD on scalars and tensor expressions that evaluate to scalars.

```
In[50]:= Tensor[ϕ] + Su[a] Sd[a]
         CovariantD[%, {b, c}]
         % // ExpandCovariantD[labs, {d, e}]
         % // TensorSimplify
         % // MapLevelParts[UpDownSwap[a], {{5, 6, 8}}]
```

$$\text{Out[50]=}\quad \phi + S^a S_a$$

$$\text{Out[51]=}\quad \phi_{;bc} + S^a{}_{;c} S_{a;b} + S^a{}_{;b} S_{a;c} + S_{a;bc} S^a + S^a{}_{;bc} S_a$$

$$\text{Out[52]=}\quad \frac{\partial^2 \phi}{\partial x^c \partial x^b} - \Gamma^e{}_{cb} \frac{\partial \phi}{\partial x^e} + \left(S^d \Gamma^a{}_{cd} + \frac{\partial S^a}{\partial x^c}\right) \left(-S_e \Gamma^e{}_{ba} + \frac{\partial S_a}{\partial x^b}\right) + \left(S^d \Gamma^a{}_{bd} + \frac{\partial S^a}{\partial x^b}\right) \left(-S_e \Gamma^e{}_{ca} + \frac{\partial S_a}{\partial x^c}\right) +$$
$$S_a \left(\frac{\partial^2 S^a}{\partial x^c \partial x^b} - \Gamma^e{}_{cb} \left(S^d \Gamma^a{}_{ed} + \frac{\partial S^a}{\partial x^e}\right) + \Gamma^a{}_{bd} \frac{\partial S^d}{\partial x^c} + \Gamma^a{}_{ce} \left(S^d \Gamma^e{}_{bd} + \frac{\partial S^e}{\partial x^b}\right) + S^d \frac{\partial \Gamma^a{}_{bd}}{\partial x^c}\right) +$$
$$S^a \left(\frac{\partial^2 S_a}{\partial x^c \partial x^b} - \Gamma^e{}_{cb} \left(-S_d \Gamma^d{}_{ea} + \frac{\partial S_a}{\partial x^e}\right) - \Gamma^d{}_{ba} \frac{\partial S_d}{\partial x^c} - \Gamma^e{}_{ca} \left(-S_d \Gamma^d{}_{be} + \frac{\partial S_e}{\partial x^b}\right) - S_d \frac{\partial \Gamma^d{}_{ba}}{\partial x^c}\right)$$

$$\text{Out[53]=}\quad S^d S_a \Gamma^a{}_{ce} \Gamma^e{}_{bd} - S^d S_e \Gamma^a{}_{bd} \Gamma^e{}_{ca} + \frac{\partial^2 \phi}{\partial x^c \partial x^b} - \Gamma^e{}_{cb} \frac{\partial \phi}{\partial x^e} + S_a \frac{\partial^2 S^a}{\partial x^c \partial x^b} -$$
$$S_a \Gamma^e{}_{cb} \frac{\partial S^a}{\partial x^e} + S^a \frac{\partial^2 S_a}{\partial x^c \partial x^b} + \frac{\partial S^a}{\partial x^c} \frac{\partial S_a}{\partial x^b} + \frac{\partial S^a}{\partial x^b} \frac{\partial S_a}{\partial x^c} - S^a \Gamma^e{}_{cb} \frac{\partial S_a}{\partial x^e}$$

$$\text{Out[54]=}\quad S^d S_a \Gamma^a{}_{ce} \Gamma^e{}_{bd} - S^d S_e \Gamma^a{}_{bd} \Gamma^e{}_{ca} + \frac{\partial^2 \phi}{\partial x^c \partial x^b} -$$
$$\Gamma^e{}_{cb} \frac{\partial \phi}{\partial x^e} + 2 S^a \frac{\partial^2 S_a}{\partial x^c \partial x^b} + 2 \frac{\partial S^a}{\partial x^b} \frac{\partial S_a}{\partial x^c} - 2 S^a \Gamma^e{}_{cb} \frac{\partial S_a}{\partial x^e}$$

We cannot expand the following covariant derivatives because they contain non tensor items.

```
In[55]:= {CovariantD[PartialD[labs][Td[a], xu[b]], c], CovariantD[Γudd[a, b, c], d]}
         % // ExpandCovariantD[labs, i]
```

$$\text{Out[55]=}\quad \left\{\left(\frac{\partial T_a}{\partial x^b}\right)_{;c}, \Gamma^a{}_{bc;d}\right\}$$

```
ExpandCovariantD::nottensor :
 A covariant derivative , {(∂Ta/∂xb);c, Γa bc;d}, cannot be expanded
   because Tensorial cannot assess the tensor nature of the expression.
```

```
Out[56]= $Aborted
```

However, by setting permissive to True, Tensorial will do the expansion. (It would be incorrect in this example.)

```
In[57]:= {CovariantD[PartialD[labs][Td[a], xu[b]], c], CovariantD[Γudd[a, b, c], d]} //
          ToFlavor[red]
         % // ExpandCovariantD[labs, red@i, True]
```

$$\text{Out[57]=}\quad \left\{\left(\frac{\partial T_a}{\partial x^b}\right)_{;c}, \Gamma^a{}_{bc;d}\right\}$$

$$\text{Out[58]=}\quad \left\{\frac{\partial^2 T_a}{\partial x^c \partial x^b} - \Gamma^i{}_{cb} \frac{\partial T_a}{\partial x^i} - \Gamma^i{}_{ca} \frac{\partial T_i}{\partial x^b}, \Gamma^a{}_{di} \Gamma^i{}_{bc} - \Gamma^a{}_{ic} \Gamma^i{}_{db} - \Gamma^a{}_{bi} \Gamma^i{}_{dc} + \frac{\partial \Gamma^a{}_{bc}}{\partial x^d}\right\}$$

The following expression won't expand at all because it contains a base index. (Take the covariant derivative of the abstract index expression and then expand to base indices.)

```
In[59]:= CovariantD[Td[1], b]
         % // ExpandCovariantD[labs, i, True]
```

Out[59]= $T_{1;b}$

```
ExpandCovariantD::nottensor :
 A covariant derivative , T1;b, cannot be expanded because
   Tensorial cannot assess the tensor nature of the expression.
```

```
Out[60]= $Aborted
```

The following expression won't expand because it contains an unexpanded partial derivative.

```
In[61]:= PartialD[Td[a], b]
         CovariantD[%, c]
         % // ExpandCovariantD[labs, i, True]
```

Out[61]= $T_{a,b}$

Out[62]= $(T_{a,b})_{;c}$

```
ExpandCovariantD::nottensor :
 A covariant derivative , (Ta,b);c, cannot be expanded because
   Tensorial cannot assess the tensor nature of the expression.
```

```
Out[63]= $Aborted
```

Expand the partial derivative first.

```
In[64]:= PartialD[Td[a], b]
         CovariantD[%, c]
         % // ExpandPartialD[labs]
         % // ExpandCovariantD[labs, i, True]
```

Out[64]= $T_{a,b}$

Out[65]= $(T_{a,b})_{;c}$

Out[66]= $\left(\frac{\partial T_a}{\partial x^b}\right)_{;c}$

Out[67]= $\frac{\partial^2 T_a}{\partial x^c \partial x^b} - \Gamma^i{}_{cb}\frac{\partial T_a}{\partial x^i} - \Gamma^i{}_{ca}\frac{\partial T_i}{\partial x^b}$

The following is a second order partial derivative of a tensor.

```
In[68]:= Td[a]
         CovariantD[%, {b, c}]
         step1 = % // ExpandCovariantD[labs, {i, j}]
```

Out[68]= T_a

Out[69]= $T_{a;bc}$

Out[70]= $\frac{\partial^2 T_a}{\partial x^c \partial x^b} - \Gamma^j{}_{cb}\left(-T_i\,\Gamma^i{}_{ja} + \frac{\partial T_a}{\partial x^j}\right) - \Gamma^i{}_{ba}\frac{\partial T_i}{\partial x^c} - \Gamma^j{}_{ca}\left(-T_i\,\Gamma^i{}_{bj} + \frac{\partial T_j}{\partial x^b}\right) - T_i\frac{\partial \Gamma^i{}_{ba}}{\partial x^c}$

The following steps perform the covariant derivative in two separate steps. The first step generates partial derivatives and Christoffels. Nevertheless, the complete expression is a valid tensor so the user would be justified in granting permission to covariantly expand.

```
In[71]:= Td[a]
         CovariantD[%, b]
         % // ExpandCovariantD[labs, i]
         Print["The expression above is a
            valid tensor even though the individual terms aren't"]
         CovariantD[%%, c]
         step2 = % // ExpandCovariantD[labs, j, True]
         Print["Check that the two answers are the same"]
         step1 - step2 // TensorSimplify
```

Out[71]= T_a

Out[72]= $T_{a;b}$

Out[73]= $-T_i \Gamma^i{}_{ba} + \frac{\partial T_a}{\partial x^b}$

```
The expression above is a valid tensor even though the individual terms aren't
```

Out[75]= $\left(\frac{\partial T_a}{\partial x^b}\right)_{;c} - \Gamma^i{}_{ba;c} T_i - T_{i;c} \Gamma^i{}_{ba}$

Out[76]= $\frac{\partial^2 T_a}{\partial x^c \partial x^b} - \Gamma^j{}_{cb} \frac{\partial T_a}{\partial x^j} - \Gamma^i{}_{ba} \left(-T_j \Gamma^j{}_{ci} + \frac{\partial T_i}{\partial x^c}\right) -$

$\Gamma^j{}_{ca} \frac{\partial T_j}{\partial x^b} - T_i \left(\Gamma^i{}_{cj} \Gamma^j{}_{ba} - \Gamma^i{}_{bj} \Gamma^j{}_{ca} - \Gamma^i{}_{ja} \Gamma^j{}_{cb} + \frac{\partial \Gamma^i{}_{ba}}{\partial x^c}\right)$

```
Check that the two answers are the same
```

Out[78]= 0

Restore settings.

```
In[79]:= ClearTensorValues /@ {gdd[i, j], guu[i, j], Γudd[i, j, k], Γddd[i, j, k], δud[i, j]};

In[80]:= ClearTensorShortcuts[{{x, y, S, T}, 1}, {{g, δ, S, T}, 2}, {{Γ, Λ, zero}, 3}]

In[81]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors, labs, step1, step2, covmn]
```

ExpandDotArray

- `ExpandDotArray[`*tensorpattern, transposeorder*`][`*expr*`]` will expand tensors that fit tensorpattern into arrays and wrap them in `MatrixForm`.

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.

The expansion is done in the sort order of the raw indices in each tensor. The lowest sort order index will be at the highest level.

The optional argument transposeorder can be used to transpose the levels of the resulting array. Arrays can either be transposed with this argument or in the `DotOperate` command.

This is intended to be used on equations that have been put in the dot mode with `DotTensorFactors`.

See the notes for `DotTensorFactors`.

See also: `DotTensorFactors`, `DotOperate`, `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}, {S, 3}]
```

The following are two examples from `DotTensorFactors`. We go one step further and expand the tensors.

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

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

```
Out[6]= R_{a c} == T_a^b . S_{b c}
```

$$\text{Out[7]= } \begin{pmatrix} R_{1\,1} & R_{1\,2} & R_{1\,3} \\ R_{2\,1} & R_{2\,2} & R_{2\,3} \\ R_{3\,1} & R_{3\,2} & R_{3\,3} \end{pmatrix} == \begin{pmatrix} T_1^{\,1} & T_1^{\,2} & T_1^{\,3} \\ T_2^{\,1} & T_2^{\,2} & T_2^{\,3} \\ T_3^{\,1} & T_3^{\,2} & T_3^{\,3} \end{pmatrix} . \begin{pmatrix} S_{1\,1} & S_{1\,2} & S_{1\,3} \\ S_{2\,1} & S_{2\,2} & S_{2\,3} \\ S_{3\,1} & S_{3\,2} & S_{3\,3} \end{pmatrix}$$

In the second example we have to expand the tensor product of e and f and not each tensor individually.

```
In[8]:= Rd[c] == 3 eu[a] fu[b] Sddd[a, b, c]
        % // MapLevelParts[DotTensorFactors[{{1, 2}, 3}], {2, {2, 3, 4}}]
        % // ExpandDotArray[Tensor[R | S, _, _] | eu[_] fu[_]]
```

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

$$\text{Out[9]=}\quad R_c == 3\, (e^a\, f^b) \cdot S_{a\,b\,c}$$

$$\text{Out[10]=}\quad \begin{pmatrix} R_1 \\ R_2 \\ R_3 \end{pmatrix} == 3 \begin{pmatrix} e^1 f^1 & e^1 f^2 & e^1 f^3 \\ e^2 f^1 & e^2 f^2 & e^2 f^3 \\ e^3 f^1 & e^3 f^2 & e^3 f^3 \end{pmatrix} \cdot \begin{pmatrix} \begin{pmatrix} S_{1\,1\,1} \\ S_{1\,1\,2} \\ S_{1\,1\,3} \end{pmatrix} & \begin{pmatrix} S_{1\,2\,1} \\ S_{1\,2\,2} \\ S_{1\,2\,3} \end{pmatrix} & \begin{pmatrix} S_{1\,3\,1} \\ S_{1\,3\,2} \\ S_{1\,3\,3} \end{pmatrix} \\ \begin{pmatrix} S_{2\,1\,1} \\ S_{2\,1\,2} \\ S_{2\,1\,3} \end{pmatrix} & \begin{pmatrix} S_{2\,2\,1} \\ S_{2\,2\,2} \\ S_{2\,2\,3} \end{pmatrix} & \begin{pmatrix} S_{2\,3\,1} \\ S_{2\,3\,2} \\ S_{2\,3\,3} \end{pmatrix} \\ \begin{pmatrix} S_{3\,1\,1} \\ S_{3\,1\,2} \\ S_{3\,1\,3} \end{pmatrix} & \begin{pmatrix} S_{3\,2\,1} \\ S_{3\,2\,2} \\ S_{3\,2\,3} \end{pmatrix} & \begin{pmatrix} S_{3\,3\,1} \\ S_{3\,3\,2} \\ S_{3\,3\,3} \end{pmatrix} \end{pmatrix}$$

In the following the S matrix expansion would normally put i at the top level and j at the lower level. But for normal array multiplication we want the i index at the lowest level. This is accomplished by transposing S when expanding.

```
In[11]:= fu[j] == Sud[j, i] eu[i]
         MapAt[DotTensorFactors[{2, 1}], %, 2]
         % // ExpandDotArray[Tensor[S, __], {2, 1}] // ExpandDotArray[Tensor[e | f, __]]
```

$$\text{Out[11]=}\quad f^j == e^i\, S^j{}_i$$

$$\text{Out[12]=}\quad f^j == S^j{}_i \cdot e^i$$

$$\text{Out[13]=}\quad \begin{pmatrix} f^1 \\ f^2 \\ f^3 \end{pmatrix} == \begin{pmatrix} S^1{}_1 & S^1{}_2 & S^1{}_3 \\ S^2{}_1 & S^2{}_2 & S^2{}_3 \\ S^3{}_1 & S^3{}_2 & S^3{}_3 \end{pmatrix} \cdot \begin{pmatrix} e^1 \\ e^2 \\ e^3 \end{pmatrix}$$

Restore settings.

```
In[14]:= ClearTensorShortcuts[{{e, f, R}, 1}, {{R, S, T}, 2}, {S, 3}]

In[15]:= DeclareBaseIndices @@ oldindices
         Clear[oldindices]
```

ExpandLieD

- `ExpandLieD[{x, δ, g, Γ}, a][expr]` will expand a first order Lie derivative of expr using coordinate positions x, Kronecker δ and dummy index a.
- `ExpandLieD[{x, δ, g, Γ}, {a, b, ...}][expr]` will expand higher order Lie derivatives using the list `{a, b, c, ...}` as dummy indices.

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

A common set of tensor labels, `{x, δ, g, Γ}`, is used in all derivatives, even though every derivative does not used every label. It will often be convenient to set a variable to the list that you are using in your application.

ExpandLieD is automatically mapped over arrays, equations and sums.

See also: `LieD`, `SetLieDisplay`, `AbsoluteD`, `CovariantD`, `PartialD`, `TotalD`.

Examples

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

Save the settings.

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

In[6]:= DefineTensorShortcuts[{{x, S, T, V, W}, 1}, {{δ, S}, 2}]
        labs = {x, δ, g, Γ};
        SetTensorValues[δud[i, j], IdentityMatrix[NDim]]
```

Set the Lie display for unexpanded Lie derivatives.

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

The Lie derivative takes on specific meaning when it is expanded to the partial derivative form.

```
In[10]:= Tu[i]
         LieD[%, V]
         % // ExpandLieD[labs, a]

Out[10]= T^i

Out[11]= £_V T^i

Out[12]= V^a ∂T^i/∂x^a - T^a ∂V^i/∂x^a
```

Lie derivative of a scalar tensor...

```
In[13]:= Tensor[ϕ]
         LieD[%, V]
         % // ExpandLieD[labs, μ]
         % // EinsteinSum[]
```

Out[13]= ϕ

Out[14]= $\pounds_V \phi$

Out[15]= $V^\mu \frac{\partial \phi}{\partial x^\mu}$

Out[16]= $V^1 \frac{\partial \phi}{\partial x^1} + V^2 \frac{\partial \phi}{\partial x^2} + V^3 \frac{\partial \phi}{\partial x^3}$

With flavored expressions, the flavor should also be on the expansion index.

```
In[17]:= Tu[i] // ToFlavor[red]
         LieD[%, V]
         % // ExpandLieD[labs, red@μ]
```

Out[17]= T^i

Out[18]= $\pounds_V T^i$

Out[19]= $V^\mu \frac{\partial T^i}{\partial x^\mu} - T^\mu \frac{\partial V^i}{\partial x^\mu}$

Higher order derivatives are also supported and require additional dummy indices.

```
In[20]:= Tu[i] // ToFlavor[red]
         LieD[%, {V, V}]
         % // ExpandLieD[labs, red /@ {μ, ν}]
```

Out[20]= T^i

Out[21]= $\pounds_{V\,V} T^i$

Out[22]= $V^\nu \left(V^\mu \frac{\partial^2 T^i}{\partial x^\nu \partial x^\mu} - T^\mu \frac{\partial^2 V^i}{\partial x^\nu \partial x^\mu} - \frac{\partial T^\mu}{\partial x^\nu} \frac{\partial V^i}{\partial x^\mu} + \frac{\partial T^i}{\partial x^\mu} \frac{\partial V^\mu}{\partial x^\nu} \right) - \frac{\partial V^i}{\partial x^\nu} \left(V^\mu \frac{\partial T^\nu}{\partial x^\mu} - T^\mu \frac{\partial V^\nu}{\partial x^\mu} \right)$

The Lie derivative of a product of two scalars.

```
In[23]:= Tensor[ϕ] Tensor[ψ]
         LieD[%, {V, V}] // HoldOp[LieD]
         % // ReleaseHold
         % // ExpandLieD[labs, {a, b}]
```

Out[23]= $\phi\, \psi$

Out[24]= $\pounds_{V\,V} (\phi\, \psi)$

Out[25]= $2\, \pounds_V \phi\, \pounds_V \psi + \pounds_{V\,V} \psi\, \phi + \pounds_{V\,V} \phi\, \psi$

Out[26]= $2\, V^a\, V^b \frac{\partial \phi}{\partial x^a} \frac{\partial \psi}{\partial x^b} + \psi\, V^b \left(V^a \frac{\partial^2 \phi}{\partial x^b \partial x^a} + \frac{\partial \phi}{\partial x^a} \frac{\partial V^a}{\partial x^b} \right) + \phi\, V^b \left(V^a \frac{\partial^2 \psi}{\partial x^b \partial x^a} + \frac{\partial \psi}{\partial x^a} \frac{\partial V^a}{\partial x^b} \right)$

Lie derivative of a product of a scalar tensor and a 2nd order tensor with respect to two different vector fields.

```
In[27]:= Tensor[ϕ] Sud[i, j] // ToFlavor[red]
         LieD[%, {V, W}]
         % // ExpandLieD[labs, red /@ {μ, ν}]
```

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

$$\text{Out[28]=}\quad £_{W}\phi\, £_{V} S^{i}{}_{j} + £_{V}\phi\, £_{W} S^{i}{}_{j} + £_{V W}\phi\, S^{i}{}_{j}$$

$$\begin{aligned}\text{Out[29]=}\quad & S^{i}{}_{j}\, W^{\nu}\left(V^{\mu}\,\frac{\partial^{2}\phi}{\partial x^{\nu}\,\partial x^{\mu}} + \frac{\partial\phi}{\partial x^{\mu}}\,\frac{\partial V^{\mu}}{\partial x^{\nu}}\right) + \\ & W^{\mu}\,\frac{\partial\phi}{\partial x^{\mu}}\left(V^{\nu}\,\frac{\partial S^{i}{}_{j}}{\partial x^{\nu}} - S^{\nu}{}_{j}\,\frac{\partial V^{i}}{\partial x^{\nu}} + S^{i}{}_{\nu}\,\frac{\partial V^{\nu}}{\partial x^{j}}\right) + V^{\mu}\,\frac{\partial\phi}{\partial x^{\mu}}\left(W^{\nu}\,\frac{\partial S^{i}{}_{j}}{\partial x^{\nu}} - S^{\nu}{}_{j}\,\frac{\partial W^{i}}{\partial x^{\nu}} + S^{i}{}_{\nu}\,\frac{\partial W^{\nu}}{\partial x^{j}}\right) + \\ & \phi\left(W^{\nu}\left(V^{\mu}\,\frac{\partial^{2} S^{i}{}_{j}}{\partial x^{\nu}\,\partial x^{\mu}} - S^{\mu}{}_{j}\,\frac{\partial^{2} V^{i}}{\partial x^{\nu}\,\partial x^{\mu}} - \frac{\partial S^{\mu}{}_{j}}{\partial x^{\nu}}\,\frac{\partial V^{i}}{\partial x^{\mu}} + S^{i}{}_{\mu}\,\frac{\partial^{2} V^{\mu}}{\partial x^{\nu}\,\partial x^{j}} + \frac{\partial S^{i}{}_{\mu}}{\partial x^{\nu}}\,\frac{\partial V^{\mu}}{\partial x^{j}} + \frac{\partial S^{i}{}_{j}}{\partial x^{\mu}}\,\frac{\partial V^{\mu}}{\partial x^{\nu}}\right) - \right. \\ & \left.\left(V^{\mu}\,\frac{\partial S^{\nu}{}_{j}}{\partial x^{\mu}} + S^{\nu}{}_{\mu}\,\frac{\partial V^{\mu}}{\partial x^{j}} - S^{\mu}{}_{j}\,\frac{\partial V^{\nu}}{\partial x^{\mu}}\right)\frac{\partial W^{i}}{\partial x^{\nu}} + \left(V^{\mu}\,\frac{\partial S^{i}{}_{\nu}}{\partial x^{\mu}} - S^{\mu}{}_{\nu}\,\frac{\partial V^{i}}{\partial x^{\mu}} + S^{i}{}_{\mu}\,\frac{\partial V^{\mu}}{\partial x^{\nu}}\right)\frac{\partial W^{\nu}}{\partial x^{j}}\right)\end{aligned}$$

ExpandLieD maps over arrays, equations and sums.

```
In[30]:= {LieD[Tensor[ϕ], V] + LieD[Tensor[ψ], W] == 0, LieD[a Tu[i], W] ≤ 0}
         % // ExpandLieD[labs, μ]
```

$$\text{Out[30]=}\quad \{£_{V}\phi + £_{W}\psi == 0,\ a\, £_{W} T^{i} \le 0\}$$

$$\text{Out[31]=}\quad \left\{V^{\mu}\,\frac{\partial\phi}{\partial x^{\mu}} + W^{\mu}\,\frac{\partial\psi}{\partial x^{\mu}} == 0,\ a\left(W^{\mu}\,\frac{\partial T^{i}}{\partial x^{\mu}} - T^{\mu}\,\frac{\partial W^{i}}{\partial x^{\mu}}\right) \le 0\right\}$$

An incorrect number of indices passes through unevaluated.

```
In[32]:= Tu[i] // ToFlavor[red]
         LieD[%, {V, V}]
         % // ExpandLieD[labs, red@μ]
```

$$\text{Out[32]=}\quad T^{i}$$

$$\text{Out[33]=}\quad £_{V V} T^{i}$$

$$\text{Out[34]=}\quad £_{V V} T^{i}$$

The expansion operates on the terms it matches.

```
In[35]:= LieD[a Tu[i], {V, V}] + LieD[b Tu[i], V] // ToFlavor[red]
         % // ExpandLieD[labs, red@μ]
```

$$\text{Out[35]=}\quad b\, £_{V} T^{i} + a\, £_{V V} T^{i}$$

$$\text{Out[36]=}\quad a\, £_{V V} T^{i} + b\left(V^{\mu}\,\frac{\partial T^{i}}{\partial x^{\mu}} - T^{\mu}\,\frac{\partial V^{i}}{\partial x^{\mu}}\right)$$

Products of sums will have to be expanded before ExpandLieD will operate.

```
In[37]:= (LieD[Tu[i], V] + LieD[Tu[i], W]) (LieD[Tu[j], V] - LieD[Tu[j], W]) == Suu[i, j]
         Print["ExpandLieD does not match without expansion"]
         %% // ExpandLieD[labs, {α, β}]
         Print["But works after expansion"]
         %% // ExpandAll
         % // ExpandLieD[labs, {α, β}]
```

$$\text{Out[37]=}\quad (\pounds_V T^i + \pounds_W T^i)\left(\pounds_V T^j - \pounds_W T^j\right) == S^{i\,j}$$

ExpandLieD does not match without expansion

$$\text{Out[39]=}\quad \left(V^\alpha \frac{\partial T^i}{\partial x^\alpha} - T^\alpha \frac{\partial V^i}{\partial x^\alpha}\right)\left(V^\beta \frac{\partial T^j}{\partial x^\beta} - T^\beta \frac{\partial V^j}{\partial x^\beta}\right) + \left(V^\beta \frac{\partial T^j}{\partial x^\beta} - T^\beta \frac{\partial V^j}{\partial x^\beta}\right)\left(W^\alpha \frac{\partial T^i}{\partial x^\alpha} - T^\alpha \frac{\partial W^i}{\partial x^\alpha}\right) -$$
$$\left(V^\alpha \frac{\partial T^i}{\partial x^\alpha} - T^\alpha \frac{\partial V^i}{\partial x^\alpha}\right)\left(W^\beta \frac{\partial T^j}{\partial x^\beta} - T^\beta \frac{\partial W^j}{\partial x^\beta}\right) - \left(W^\alpha \frac{\partial T^i}{\partial x^\alpha} - T^\alpha \frac{\partial W^i}{\partial x^\alpha}\right)\left(W^\beta \frac{\partial T^j}{\partial x^\beta} - T^\beta \frac{\partial W^j}{\partial x^\beta}\right) == S^{i\,j}$$

But works after expansion

$$\begin{aligned}\text{Out[41]=}\quad & V^\alpha V^\beta \frac{\partial T^i}{\partial x^\alpha}\frac{\partial T^j}{\partial x^\beta} + V^\beta W^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial T^j}{\partial x^\beta} - V^\alpha W^\beta \frac{\partial T^i}{\partial x^\alpha}\frac{\partial T^j}{\partial x^\beta} - W^\alpha W^\beta \frac{\partial T^i}{\partial x^\alpha}\frac{\partial T^j}{\partial x^\beta} - \\ & T^\alpha V^\beta \frac{\partial T^j}{\partial x^\beta}\frac{\partial V^i}{\partial x^\alpha} + T^\alpha W^\beta \frac{\partial T^j}{\partial x^\beta}\frac{\partial V^i}{\partial x^\alpha} - T^\beta V^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial V^j}{\partial x^\beta} - T^\beta W^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial V^j}{\partial x^\beta} + \\ & T^\alpha T^\beta \frac{\partial V^i}{\partial x^\alpha}\frac{\partial V^j}{\partial x^\beta} - T^\alpha V^\beta \frac{\partial T^j}{\partial x^\beta}\frac{\partial W^i}{\partial x^\alpha} + T^\alpha W^\beta \frac{\partial T^j}{\partial x^\beta}\frac{\partial W^i}{\partial x^\alpha} + T^\alpha T^\beta \frac{\partial V^j}{\partial x^\beta}\frac{\partial W^i}{\partial x^\alpha} + \\ & T^\beta V^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial W^j}{\partial x^\beta} + T^\beta W^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial W^j}{\partial x^\beta} - T^\alpha T^\beta \frac{\partial V^i}{\partial x^\alpha}\frac{\partial W^j}{\partial x^\beta} - T^\alpha T^\beta \frac{\partial W^i}{\partial x^\alpha}\frac{\partial W^j}{\partial x^\beta} == S^{i\,j}\end{aligned}$$

$$\begin{aligned}\text{Out[42]=}\quad & V^\alpha V^\beta \frac{\partial T^i}{\partial x^\alpha}\frac{\partial T^j}{\partial x^\beta} + V^\beta W^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial T^j}{\partial x^\beta} - V^\alpha W^\beta \frac{\partial T^i}{\partial x^\alpha}\frac{\partial T^j}{\partial x^\beta} - W^\alpha W^\beta \frac{\partial T^i}{\partial x^\alpha}\frac{\partial T^j}{\partial x^\beta} - \\ & T^\alpha V^\beta \frac{\partial T^j}{\partial x^\beta}\frac{\partial V^i}{\partial x^\alpha} + T^\alpha W^\beta \frac{\partial T^j}{\partial x^\beta}\frac{\partial V^i}{\partial x^\alpha} - T^\beta V^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial V^j}{\partial x^\beta} - T^\beta W^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial V^j}{\partial x^\beta} + \\ & T^\alpha T^\beta \frac{\partial V^i}{\partial x^\alpha}\frac{\partial V^j}{\partial x^\beta} - T^\alpha V^\beta \frac{\partial T^j}{\partial x^\beta}\frac{\partial W^i}{\partial x^\alpha} + T^\alpha W^\beta \frac{\partial T^j}{\partial x^\beta}\frac{\partial W^i}{\partial x^\alpha} + T^\alpha T^\beta \frac{\partial V^j}{\partial x^\beta}\frac{\partial W^i}{\partial x^\alpha} + \\ & T^\beta V^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial W^j}{\partial x^\beta} + T^\beta W^\alpha \frac{\partial T^i}{\partial x^\alpha}\frac{\partial W^j}{\partial x^\beta} - T^\alpha T^\beta \frac{\partial V^i}{\partial x^\alpha}\frac{\partial W^j}{\partial x^\beta} - T^\alpha T^\beta \frac{\partial W^i}{\partial x^\alpha}\frac{\partial W^j}{\partial x^\beta} == S^{i\,j}\end{aligned}$$

Restore the settings.

```
In[43]:= ClearTensorShortcuts[{{x, S, T, V, W}, 1}, {{δ, S}, 2}]

In[44]:= SetLieDisplay["PlainMode"]

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

ExpandPartialD

- `ExpandPartialD[{x, δ, g, Γ}][expr]` expands the partial derivatives using the coordinate label x and the Kronecker label δ.

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.

ExpandPartialD is just a delayed form of `PartialD[labs]` and it does not directly allow you to introduce partial derivatives with respect to symbols.

In `ExpandPartialD` 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 `ExpandPartialD`.

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

See also: `PartialD`, `NondependentPartialD`, `TotalD`, `CovariantD`, `AbsoluteD`.

Examples

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

Save settings.

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

In[5]:= DefineTensorShortcuts[{{x, T}, 1}, {{S, T, δ}, 2}]
```

Set the standard labels to be used in the examples.

```
In[6]:= labs = {x, δ, g, Γ};
```

Without information as to the coordinate positions, a partial derivative simply shows a comma before the differentiated indices. These expressions can be expanded to explicit partial derivatives by using `ExpandPartialD` and supplying the coordinate and Kronecker labels.

```
In[7]:= Td[i]
        PartialD[%, j]
        % // ExpandPartialD[labs]

Out[7]= T_i

Out[8]= T_{i,j}

Out[9]= ∂T_i / ∂x^j
```

For flavored expressions, the flavor must also be on the derivative index, but nothing further is needed in expanding.

```
In[10]:= Td[i] + xu[i] // ToFlavor[red]
         PartialD[%, red@j]
         % // ExpandPartialD[labs]
```

$$\text{Out[10]=}\quad T_i + x^i$$

$$\text{Out[11]=}\quad T_{i,j} + x^i{}_{,j}$$

$$\text{Out[12]=}\quad \delta^i{}_j + \frac{\partial T_i}{\partial x^j}$$

All the tensors containing partial derivatives are expanded.

```
In[13]:= {xu[i] + Tu[i], 2 xu[i] Tu[j], Tu[i] Sud[p, i]} // ToFlavor[red]
         PartialD[#, red@k] & /@ %
         % // ExpandPartialD[labs]
```

$$\text{Out[13]=}\quad \left\{T^i + x^i,\ 2\,T^j\,x^i,\ S^p{}_i\,T^i\right\}$$

$$\text{Out[14]=}\quad \left\{T^i{}_{,k} + x^i{}_{,k},\ 2\left(x^i{}_{,k}\,T^j + T^j{}_{,k}\,x^i\right),\ T^i{}_{,k}\,S^p{}_i + S^p{}_{i,k}\,T^i\right\}$$

$$\text{Out[15]=}\quad \left\{\delta^i{}_k + \frac{\partial T^i}{\partial x^k},\ 2\left(T^j\,\delta^i{}_k + x^i\,\frac{\partial T^j}{\partial x^k}\right),\ T^i\,\frac{\partial S^p{}_i}{\partial x^k} + S^p{}_i\,\frac{\partial T^i}{\partial x^k}\right\}$$

Restore settings.

```
In[16]:= ClearTensorShortcuts[{{x, T}, 1}, {{S, T, δ}, 2}]
```

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

ExpandTotalD

- `ExpandTotalD[{x, δ, g, Γ}, a][expr]` expands total derivative expressions with coordinate label x and Kronecker label δ, using a as the dummy index.
- `ExpandTotalD[{x, δ, g, Γ}, {a, b, ...}][expr]` expands higher order derivatives.

Tensors are funtions of the coordinate positions and any variation of the tensor is due to the variation of the coordiates over the parameter of differentiation used. Tensors are not allowed to depend on the parameters directly but only through the coordinates.!

For multiple differentiations, a dummy index for each differentiation must be given.

The dummy indices must be in the desired flavor.

A common set of tensor labels, `{x, δ, g, Γ}`, is used in all derivatives, even though every derivative does not used every label. It will often be convenient to set a variable to the list that you are using in your application.

ExpandTotalD is mapped over arrays, equations and sums.

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

See also: `TotalD`, `CovariantD`, `PartialD`, `AbsoluteD`, `ExpandPartialD`.

Examples

We set the Euclidean metric and coordinates.

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

Save the settings.

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

In[6]:= labs = {x, δ, g, Γ};
        DefineTensorShortcuts[{{x, F, G}, 1}, {{g, δ}, 2}]
        SetTensorValues[δud[i, j], IdentityMatrix[3]]
```

Here is the total derivative of a tensor F...

```
In[9]:= Fu[i]
        TotalD[%, t]

Out[9]= F^i

Out[10]= dF^i/dt
```

The actual meaning of `TotalD` is...

```
In[11]:= Fu[i]
         TotalD[%, t]
         % // ExpandTotalD[labs, a]
```

Out[11]= F^i

Out[12]= $\frac{dF^i}{dt}$

Out[13]= $\frac{dx^a}{dt}\frac{\partial F^i}{\partial x^a}$

ExpandTotalD is mapped over arrays, equations and sums.

```
In[14]:= TotalD[{Fu[i] + Gu[i], Fu[j]}, t] == {0, 0} // Thread;
         MapThread[#2 @@ #1 &, {%, {Equal, LessEqual}}]
         % // ExpandTotalD[labs, a]
```

Out[15]= $\left\{\frac{dF^i}{dt} + \frac{dG^i}{dt} == 0,\ \frac{dF^j}{dt} \le 0\right\}$

Out[16]= $\left\{\frac{dx^a}{dt}\frac{\partial F^i}{\partial x^a} + \frac{dx^a}{dt}\frac{\partial G^i}{\partial x^a} == 0,\ \frac{dx^a}{dt}\frac{\partial F^j}{\partial x^a} \le 0\right\}$

```
In[17]:= Fu[i] Gu[j] == c
         TotalD[%, t]
         % // ExpandTotalD[labs, a]
```

Out[17]= $F^i\,G^j == c$

Out[18]= $G^j\frac{dF^i}{dt} + F^i\frac{dG^j}{dt} == \frac{dc}{dt}$

Out[19]= $G^j\frac{dx^a}{dt}\frac{\partial F^i}{\partial x^a} + F^i\frac{dx^a}{dt}\frac{\partial G^j}{\partial x^a} == \frac{dc}{dt}$

Here is a second order differentiation. For a flavored expression, the dummy indices must be in the desired flavor.

```
In[20]:= Fu[i] // ToFlavor[red]
         TotalD[%, {t, t}]
         % // ExpandTotalD[labs, red /@ {a, b}]
```

Out[20]= F^i

Out[21]= $\frac{d^2F^i}{dt\,dt}$

Out[22]= $\frac{dx^a}{dt}\frac{dx^b}{dt}\frac{\partial^2 F^i}{\partial x^b\,\partial x^a} + \frac{d^2x^a}{dt\,dt}\frac{\partial F^i}{\partial x^a}$

```
In[23]:= TotalD[Fu[i], t] TotalD[Gu[j], t] == 0
         % // ExpandTotalD[labs, {a, b}]
```

Out[23]= $\frac{dF^i}{dt}\frac{dG^j}{dt} == 0$

Out[24]= $\frac{dx^a}{dt}\frac{dx^b}{dt}\frac{\partial F^i}{\partial x^a}\frac{\partial G^j}{\partial x^b} == 0$

```
In[25]:= (TotalD[Fu[i], t] + TotalD[Gu[i], t]) (TotalD[Fu[j], t] - TotalD[Gu[j], t]) == 0
         % // ExpandTotalD[labs, {a, b}]
```

$$\text{Out[25]=} \quad \left(\frac{dF^i}{dt} + \frac{dG^i}{dt}\right)\left(\frac{dF^j}{dt} - \frac{dG^j}{dt}\right) == 0$$

$$\text{Out[26]=} \quad \left(\frac{dx^a}{dt}\frac{\partial F^i}{\partial x^a} + \frac{dx^a}{dt}\frac{\partial G^i}{\partial x^a}\right)\left(\frac{dx^b}{dt}\frac{\partial F^j}{\partial x^b} - \frac{dx^b}{dt}\frac{\partial G^j}{\partial x^b}\right) == 0$$

The number of dummy indices must be equal to the differentiation order.

```
In[27]:= Fu[i] // ToFlavor[red]
         TotalD[%, {t, t}]
         % // ExpandTotalD[labs, red /@ {a, b, c}]
```

$$\text{Out[27]=} \quad F^i$$

$$\text{Out[28]=} \quad \frac{d^2 F^i}{dt\, dt}$$

```
ExpandTotalD::dummies : Number of dummies {a, b, c}
   does not match number of differentiation variables {t, t}
```

```
Out[29]= $Aborted
```

Here, we calculate the acceleration of a particle moving on a circular path.

```
In[30]:= SetTensorValueRules[xu[i], {Sin[t], Cos[t], 0}]
         TotalD[xu[i], {t, t}]
         % // ExpandTotalD[labs, {a, b}]
         % // KroneckerAbsorb[δ]
         % // ToArrayValues[]
```

$$\text{Out[31]=} \quad \frac{d^2 x^i}{dt\, dt}$$

$$\text{Out[32]=} \quad \delta^i{}_a \frac{d^2 x^a}{dt\, dt}$$

$$\text{Out[33]=} \quad \frac{d^2 x^i}{dt\, dt}$$

```
Out[34]= {-Sin[t], -Cos[t], 0}
```

A total derivative of a partial derivatice of a tensor. We must expand the partial derivative before taking the total derivative.

```
In[35]:= ClearTensorValues[xu[i]]
         Fu[k]
         PartialD[%, m]
         ExpandPartialD[labs][%]
         TotalD[%, t]
         ExpandTotalD[labs, i][%]
```

Out[36]= F^k

Out[37]= $F^k{}_{,m}$

Out[38]= $\frac{\partial F^k}{\partial x^m}$

Out[39]= $\frac{d\frac{\partial F^k}{\partial x^m}}{dt}$

Out[40]= $\frac{dx^i}{dt}\ \frac{\partial^2 F^k}{\partial x^i\ \partial x^m}$

If we use a partial derivative after a total derivative, the total derivative must be expanded and the corresponding expanded form of `PartialD` must be used.

```
In[41]:= Fu[k]
         TotalD[%, t]
         % // ExpandTotalD[labs, i]
         PartialD[labs][%, xu[m]]
```

Out[41]= F^k

Out[42]= $\frac{dF^k}{dt}$

Out[43]= $\frac{dx^i}{dt}\ \frac{\partial F^k}{\partial x^i}$

Out[44]= $\frac{dx^i}{dt}\ \frac{\partial^2 F^k}{\partial x^m\ \partial x^i}$

Restore the sesttings

```
In[45]:= ClearTensorValues /@ {xu[i], δud[i, j]};
         ClearTensorShortcuts[{{x, F}, 1}, {{g, δ}, 2}]
```

```
In[47]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

ExtractFreeIndices

- `ExtractFreeIndices[`*expression*`]` will return `{upindices, downindices}` where each is a list of the free indices in the expression.

The expression may be an equation, a sum of terms, or a single tensor term.

The routine may be used to check index balance in an equation or expression. If the terms don't all have the same free indices an error message is generated and False is returned.

This routine is primarily used in programming other routines.

Free indices are indices that appear only once in each term.

upindices and downindices are each returned in the natural sort order.

`ExtractFreeIndices` uses the `ParseTermIndices` routine. You can use `IndexParsingRules` to implement tensor expressions that are not normally recognized by `ParseTermIndices`.

See also: `ParseTermIndices`, `IndexParsingRules`, `EinsteinArray`.

Examples

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

Save the old settings.

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

```
In[5]:= DeclareBaseIndices[Range[3]];
        DeclareIndexFlavor /@ {{red, Red}, {rocket, SuperDagger}};
        DefineTensorShortcuts[{{x, y, F, T}, 1}, {{S, T}, 2}]
```

With an equation...

```
In[8]:= Sud[i, j] xu[j] yd[m] == Tud[i, k] xu[k] yd[m] // ToFlavor[red]
        % // ExtractFreeIndices
```

```
Out[8]= S^i_j x^j y_m == T^i_k x^k y_m
```

```
Out[9]= {{i}, {m}}
```

With a sum...

```
In[10]:= Sud[i, j] xu[j] yd[m] - Tud[i, k] xu[k] yd[m] // ToFlavor[rocket]
         % // ExtractFreeIndices
```

```
Out[10]= S^i†_j† x^j† y_m† - T^i†_k† x^k† y_m†
```

```
Out[11]= {{i†}, {m†}}
```

The following expressions are incorrect.

```
In[12]:= Sud[i, j] xu[j] yd[m] == Tud[i, k] xu[k] yd[n] // ToFlavor[red]
         % // ExtractFreeIndices
```

Out[12]= $S^{i}{}_{j}\, x^{j}\, y_{m} == T^{i}{}_{k}\, x^{k}\, y_{n}$

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

```
Out[13]= False
```

```
In[14]:= Sud[i, j] xu[j] yd[m] == Tud[i, k] xu[k] yd[k] // ToFlavor[red]
         % // ExtractFreeIndices
```

Out[14]= $S^{i}{}_{j}\, x^{j}\, y_{m} == T^{i}{}_{k}\, x^{k}\, y_{k}$

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

```
Out[15]= False
```

Expressions involving differentiations.

```
In[16]:= Sud[i, j] xu[j] CovariantD[Td[μ], m] == Tud[i, k] xu[k] CovariantD[Td[μ], m] //
          ToFlavor[red]
         % // ExtractFreeIndices
```

Out[16]= $T_{\mu;m}\, S^{i}{}_{j}\, x^{j} == T_{\mu;m}\, T^{i}{}_{k}\, x^{k}$

```
Out[17]= {{i}, {m, μ}}
```

```
In[18]:= Sud[i, j] xu[j] CovariantD[Td[μ], m] == Tud[i, k] xu[k] CovariantD[Td[μ], m] //
          ToFlavor[red]
         ExpandCovariantD[{x, δ, g, Γ}, red@a] /@ %
         % // ExtractFreeIndices
```

Out[18]= $T_{\mu;m}\, S^{i}{}_{j}\, x^{j} == T_{\mu;m}\, T^{i}{}_{k}\, x^{k}$

Out[19]= $S^{i}{}_{j}\, x^{j} \left(-T_{a}\, \Gamma^{a}{}_{m\mu} + \frac{\partial T_{\mu}}{\partial x^{m}}\right) == T^{i}{}_{k}\, x^{k} \left(-T_{a}\, \Gamma^{a}{}_{m\mu} + \frac{\partial T_{\mu}}{\partial x^{m}}\right)$

```
Out[20]= {{i}, {m, μ}}
```

```
In[21]:= Fu[i] == m TotalD[xu[i], {t, t}]
         % // ExtractFreeIndices
```

Out[21]= $F^{i} == m \frac{d^{2} x^{i}}{dt\, dt}$

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

Restore the initial settings...

```
In[23]:= ClearTensorShortcuts[{{x, y, F}, 1}, {{S, T}, 2}]
```

```
In[24]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]
```

ExtractTensorSlots & SlotsToTensor

- `ExtractTensorSlots[`*Tensor*`[`*label, ups, downs*`]]` will extract the slots of the tensor.
- `SlotsToTensor[`*slots*`]` will convert the slots obtained with ExtractTensorSlots back to a tensor.

The slots are returned as a list of the form `{{label, index, ∓1}` where -1 is used for an up index and +1 is used for a down index.

`ExtractTensorSlots` is primarily used internally but is provided to the user as a convenience.

See also: `ExtractTermSlots`.

Examples

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

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

In[3]:= testlist = {Rdd[a, b], Rud[a, b], Rud[a, a], Tuddd[a, b, c, d]};
        Thread[testlist → ExtractTensorSlots /@ testlist] // TableForm
```

```
Out[4]//TableForm=
      R_{a b} → {{R, a, 1}, {R, b, 1}}
      R^a_b → {{R, a, -1}, {R, b, 1}}
      R^a_a → {{R, a, -1}, {R, a, 1}}
      T^a_{b c d} → {{T, a, -1}, {T, b, 1}, {T, c, 1}, {T, d, 1}}
```

```
In[5]:= Tuudd[a, b, c, d]
        % // ExtractTensorSlots
        % // SlotsToTensor

Out[5]= T^{a b}_{c d}

Out[6]= {{T, a, -1}, {T, b, -1}, {T, c, 1}, {T, d, 1}}

Out[7]= T^{a b}_{c d}

In[8]:= ClearTensorShortcuts[{R, 2}, {T, 4}]

In[9]:= Clear[testlist]
```

ExtractTermSlots

■ `ExtractTermSlots[`*term*`]` will extract the slots of the tensor term in the *Mathematica* order of the factors.

The slots are returned in the form: `{scalarfactor, factorslots ..}`

The factor slots are returned as a list of the form `{{label, index, ∓1}` where -1 is used for an up index and +1 is used for a down index.

Tensors and indices in derivatives are extracted from the derivatives and the derivative information is lost.

`ExtractTermSlots` is primarily used internally but is provided to the user as a convenience.

See also: `ExtractTensorSlots`.

Examples

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

In[2]:= DefineTensorShortcuts[{x, 1}, {R, 2}, {R, 4}]
        labs99 = {x, δ, g, Γ};
```

Some examples of extracting the slot structure for terms...

```
In[4]:= testcase = {Rdd[b, a], -2 a Sin[θ] Rdd[b, a],
           Rdd[a, b] Ruudd[a, b, c, d], 2 q Rdd[a, b] Ruudd[a, b, c, d],
           Ruu[a, b] Rdddd[a, b, c, d], Ruu[c, d] Rdddd[a, b, c, d], 3};
        Thread[testcase → ExtractTermSlots /@ testcase] // TableForm
```

```
Out[5]//TableForm=
R_ba → {1, {{{R, b, 1}, {R, a, 1}}}}
-2 a Sin[θ] R_ba → {-2 a Sin[θ], {{{R, b, 1}, {R, a, 1}}}}
R_ab R^ab_cd → {1, {{{R, a, 1}, {R, b, 1}}, {{R, a, -1}, {R, b, -1}, {R, c, 1}, {R, d, 1}}}}
2 q R_ab R^ab_cd → {2 q, {{{R, a, 1}, {R, b, 1}}, {{R, a, -1}, {R, b, -1}, {R, c, 1}, {R, d, 1}}}}
R^ab R_abcd → {1, {{{R, a, -1}, {R, b, -1}}, {{R, a, 1}, {R, b, 1}, {R, c, 1}, {R, d, 1}}}}
R^cd R_abcd → {1, {{{R, c, -1}, {R, d, -1}}, {{R, a, 1}, {R, b, 1}, {R, c, 1}, {R, d, 1}}}}
3 → {1, {}}
```

```
In[6]:= testcase = {PartialD[Rdd[a, b], c],
           PartialD[labs99][Rud[a, b], xu[c]], CovariantD[Rud[a, b], c]};
        Thread[testcase → ExtractTermSlots /@ testcase] // TableForm
```

```
Out[7]//TableForm=
R_ab,c → {1, {{{R, a, 1}, {R, b, 1}}, {{tlab$87, c, 1}}}}
∂R^a_b/∂x^c → {1, {{{R, a, -1}, {R, b, 1}}, {{x, c, 1}}}}
R^a_b;c → {1, {{{R, a, -1}, {R, b, 1}}, {{tlab$90, c, 1}}}}
```

```
In[8]:= ClearTensorShortcuts[{R, 2}, {T, 4}]
        Clear[testcase, labs99]
```