CalculateChristoffels

- `CalculateChristoffels[{x, δ, g, Γ}, flavor : Identity, simplification : Identity]` calculates the down and up Christoffel symbol arrays.

The routine returns the `{Γddd, Γudd}` arrays.

`{x, δ, g, Γ}` are the same symbols used in the various derivative expansion routines. `x` is the coordinate, `δ` the Kronecker symbol, `g` the metric tensor label. `Γ` is not actually used in the routine.

A shortcut must be defined for the metric tensor g and values must have been set for its components and the inverse components.

The optional argument flavor is the flavor in which the calculation will be performed. Several metric tensors and sets of Christoffel symbols may exist at the same time.

Simplification is an optional argument that supplies a routine to simplify the Christoffel symbols. By default no simplification is done.

See also: `DeclareBaseIndices`, `SetTensorValues`, `SetTensorValueRules`, `CoordinatesToTensors`.

Examples

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

Save settings

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}, {green, SapGreen}];
        DeclareBaseIndices[{1, 2, 3}, {red, {r, ϕ, θ}}, {green, {θ, ϕ}}]
        labs = {x, δ, g, Γ};
```

```
In[8]:= DefineTensorShortcuts[{{x}, 1}, {{g}, 2}, {{Γ}, 3}]
```

The metric for a spherical coordinate system using `{1, 2, 3}` as base indices is...

```
In[9]:= (metric = DiagonalMatrix[{1, r^2, r^2 Sin[ϕ]^2}] // CoordinatesToTensors[{r, ϕ, θ}]) //
         MatrixForm
        MapThread[SetTensorValueRules[#1, #2] &,
          {{gdd[a, b], guu[a, b]}, {metric, Inverse[metric]}}];
```

```
Out[9]//MatrixForm=
⎛1  0       0                  ⎞
⎜                              ⎟
⎜0  (x[1])^2  0                ⎟
⎜                              ⎟
⎝0  0       Sin[x[2]]^2 (x[1])^2⎠
```

Calculating the up and down Christoffel symbols gives...

```
In[11]:= {downΓ, upΓ} = CalculateChristoffels[labs]
```

```
Out[11]= {{{{0, 0, 0}, {0, -x^1, 0}, {0, 0, -Sin[x^2]^2 x^1}},
    {{0, x^1, 0}, {x^1, 0, 0}, {0, 0, -Cos[x^2] Sin[x^2] (x^1)^2}}, {{0, 0, Sin[x^2]^2 x^1},
    {0, 0, Cos[x^2] Sin[x^2] (x^1)^2}, {Sin[x^2]^2 x^1, Cos[x^2] Sin[x^2] (x^1)^2, 0}}},
  {{{0, 0, 0}, {0, -x^1, 0}, {0, 0, -Sin[x^2]^2 x^1}},
    {{0, (x^1)^-1, 0}, {(x^1)^-1, 0, 0}, {0, 0, -Cos[x^2] Sin[x^2]}},
    {{0, 0, (x^1)^-1}, {0, 0, Cot[x^2]}, {(x^1)^-1, Cot[x^2], 0}}}}
```

We set value rules and display the independent nonzero Christoffel values.

```
In[12]:= SetTensorValueRules[Γddd[a, b, c], downΓ]
         SetTensorValueRules[Γudd[a, b, c], upΓ]
         SelectedTensorRules[Γ, Γudd[_, b_, c_] /; OrderedQ[{b, c}]] //
           UseCoordinates[{r, ϕ, θ}] // TableForm
```

```
Out[14]//TableForm=
Γ^1_{2 2} → -r
Γ^1_{3 3} → -r Sin[ϕ]^2
Γ^2_{1 2} → 1/r
Γ^2_{3 3} → -Cos[ϕ] Sin[ϕ]
Γ^3_{1 3} → 1/r
Γ^3_{2 3} → Cot[ϕ]
```

The following is the same metric in the red flavor that uses symbols for the base indices

```
In[15]:= (metric = DiagonalMatrix[{1, r^2, r^2 Sin[ϕ]^2}] //
            CoordinatesToTensors[{r, ϕ, θ}, x, red]) // MatrixForm
         MapThread[SetTensorValueRules[#1, #2] &,
           {{gdd[a, b], guu[a, b]} // ToFlavor[red], {metric, Inverse[metric]}}];
```

```
Out[15]//MatrixForm=
⎛1   0          0                  ⎞
⎜0   (x^r)^2    0                  ⎟
⎝0   0          Sin[x^ϕ]^2 (x^r)^2 ⎠
```

Calculating and setting rules for the Christoffel values...

```
In[17]:= {downΓr, upΓr} = CalculateChristoffels[labs, red];
         MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]} // ToFlavor[red], {downΓr, upΓr}}];
```

The Christoffel symbols now display with symbolic indices.

```
In[19]:= SelectedTensorRules[Γ, Γudd[red[_], b_, c_] /; OrderedQ[{b, c}]] //
           UseCoordinates[{r, ϕ, θ}, x, red] // TableForm
```

Out[19]//TableForm=

$\Gamma^{r}_{\phi\,\phi} \to -r$

$\Gamma^{r}_{\theta\,\theta} \to -r\,\text{Sin}[\phi]^2$

$\Gamma^{\phi}_{r\,\phi} \to \frac{1}{r}$

$\Gamma^{\phi}_{\theta\,\theta} \to -\text{Cos}[\phi]\,\text{Sin}[\phi]$

$\Gamma^{\theta}_{r\,\theta} \to \frac{1}{r}$

$\Gamma^{\theta}_{\theta\,\phi} \to \text{Cot}[\phi]$

The following is the metric on the surface of a sphere of radius `a`.

```
In[20]:= SetAttributes[a, Constant];
         (metric = DiagonalMatrix[{a^2, a^2 Sin[θ]^2}] //
            CoordinatesToTensors[{θ, ϕ}, x, green]) // MatrixForm
         MapThread[SetTensorValueRules[#1, #2] &,
           {{gdd[a, b], guu[a, b]} // ToFlavor[green], {metric, Inverse[metric]}}];
```

Out[21]//MatrixForm=

$$\begin{pmatrix} a^2 & 0 \\ 0 & a^2\,\text{Sin}[x^{\theta}]^2 \end{pmatrix}$$

In this case we specify that `Simplify` is to be used on each of the Christoffel values.

```
In[23]:= {downΓg, upΓg} = CalculateChristoffels[labs, green, Simplify];
         MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]} // ToFlavor[green], {downΓg, upΓg}}];
```

The result of using Simplify was to generate the double angle `Sin[2 θ] / 2` term instead of a `Sin[θ] Cos[θ]`.

```
In[25]:= SelectedTensorRules[Γ, Γudd[green[_], b_, c_] /; OrderedQ[{b, c}]] //
           UseCoordinates[{θ, ϕ}, x, green] // TableForm
```

Out[25]//TableForm=

$\Gamma^{\theta}_{\phi\,\phi} \to -\frac{1}{2}\,\text{Sin}[2\,\theta]$

$\Gamma^{\phi}_{\theta\,\phi} \to \text{Cot}[\theta]$

All of these Christoffel values are simultaneously available. Here they are displayed as they are actually stored, without using `UseCoordinates`.

```
In[26]:= SelectedTensorRules[Γ, Γudd[_, b_, c_] /; OrderedQ[{b, c}]] // TableForm
```

Out[26]//TableForm=

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

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

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

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

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

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

$$\Gamma^{r}{}_{\phi\,\phi} \to -x^{r}$$

$$\Gamma^{r}{}_{\theta\,\theta} \to -\text{Sin}[x^{\phi}]^{2}\, x^{r}$$

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

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

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

$$\Gamma^{\theta}{}_{\theta\,\phi} \to \text{Cot}[x^{\phi}]$$

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

$$\Gamma^{\phi}{}_{\theta\,\phi} \to \text{Cot}[x^{\theta}]$$

Restore settings

```
In[27]:= {{guu[i, j], gdd[i, j], Γddd[a, b, c], Γudd[a, b, c]},
           {guu[i, j], gdd[i, j], Γddd[a, b, c], Γudd[a, b, c]} // ToFlavor[red],
           {guu[i, j], gdd[i, j], Γddd[a, b, c], Γudd[a, b, c]} // ToFlavor[green]} // Flatten;
         ClearTensorValues /@ %;

In[29]:= ClearTensorShortcuts[{{x}, 1}, {{g}, 2}]

In[30]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors, upΓ, downΓ, upΓr, downΓr, upΓg, downΓg]
```

CalculateRiemannd

- `CalculateRiemannd[{x, g, δ, Γ}, flavor : Identity, simplifyroutine : Identity]` will calculate the down version of the Riemann tensor and return it as an array.

`x`, δ, `g` and `Γ` are the symbols for the coordinates, the Kronecker, the metric tensor and Christoffel connections. `x`, `g` and `Γ` must have defined shortcuts and `g` and `Γ` musthave been given tensor values or rules.

The optional index flavor is the flavor used in the `g` and `Γ` tensors. The optional argument simplifyroutine specifies the simplify routine to be applied to each independent element.

Only the independent elements are separately calculated and the entire array is assembled from them. Hence this will be considerable faster than generating the components from a definitional formula.

The signs of elements obtained depends on the sign used for the line element and the metric matrix. The usual practice is to use the spacelike convention where the space elements are positive and the time element is negative. Changing the sign of g changes the sign of various objects depending on how many times g appears in the definition. Down Christoffel symbols are changed but up Christoffel symbols are not. Down Riemann elements are changed but up Riemann elements are not.

See also: `CalculateRRRG`, `SetTensorValues`, `SetTensorValuesRules`, `SelectedTensorRules`.

Example

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

Save the settings and declare base indices and flavors.

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

The following calculation is from Appendix B in the Hartle *Gravitation* book.

```
In[5]:= varnames = {t, r, θ, ϕ};
        DeclareBaseIndices[varnames]
        DeclareIndexFlavor[{red, Red}]
        labs = {x, δ, g, Γ};
        DefineTensorShortcuts[{{x, e}, 1}, {{g, δ}, 2}, {Γ, 3}, {R, 4}]
```

We will calculate the Riemann tensor for the Friedmann-Robertson-Walker geometry. The metric for proper distances is...

```
In[10]:= SetAttributes[k, Constant];
         diagmetric = {-1, a[t]^2 {1/(1 - k r^2), r^2 {1, Sin[θ]^2}}} // Flatten;
         (cmetric = DiagonalMatrix[diagmetric]) // MatrixForm
         metric = % // CoordinatesToTensors[varnames];
         MapThread[SetTensorValueRules[#1, #2] &,
           {{gdd[a, b], guu[a, b]}, {metric, Inverse[metric]}}];
```

Out[12]//MatrixForm=

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & \frac{a[t]^2}{1-k\,r^2} & 0 & 0 \\ 0 & 0 & r^2\,a[t]^2 & 0 \\ 0 & 0 & 0 & r^2\,a[t]^2\,\mathrm{Sin}[\theta]^2 \end{pmatrix}$$

Next, calculate and set the Christoffel symbols.

```
In[15]:= {downΓ, upΓ} = CalculateChristoffels[labs];
         MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]}, {downΓ, upΓ}}];
         SelectedTensorRules[Γ, Γudd[_, j_, k_] /; OrderedQ[{j, k}]] //
           UseCoordinates[varnames] // TableForm
```

Out[17]//TableForm=

$$\Gamma^{t}{}_{r\,r} \to \frac{a[t]\,a'[t]}{1-k\,r^2}$$
$$\Gamma^{t}{}_{\theta\,\theta} \to r^2\,a[t]\,a'[t]$$
$$\Gamma^{t}{}_{\phi\,\phi} \to r^2\,a[t]\,\mathrm{Sin}[\theta]^2\,a'[t]$$
$$\Gamma^{r}{}_{r\,t} \to \frac{a'[t]}{a[t]}$$
$$\Gamma^{r}{}_{r\,r} \to \frac{k\,r}{1-k\,r^2}$$
$$\Gamma^{r}{}_{\theta\,\theta} \to -r\,(1-k\,r^2)$$
$$\Gamma^{r}{}_{\phi\,\phi} \to -r\,(1-k\,r^2)\,\mathrm{Sin}[\theta]^2$$
$$\Gamma^{\theta}{}_{t\,\theta} \to \frac{a'[t]}{a[t]}$$
$$\Gamma^{\theta}{}_{r\,\theta} \to \frac{1}{r}$$
$$\Gamma^{\theta}{}_{\phi\,\phi} \to -\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]$$
$$\Gamma^{\phi}{}_{t\,\phi} \to \frac{a'[t]}{a[t]}$$
$$\Gamma^{\phi}{}_{r\,\phi} \to \frac{1}{r}$$
$$\Gamma^{\phi}{}_{\theta\,\phi} \to \mathrm{Cot}[\theta]$$

We can then calculate the down form of the Riemann matrix.

```
In[18]:= riemannd = CalculateRiemannd[labs];
         SetTensorValueRules[Rdddd[μ, ν, ρ, σ], riemannd]
         (SelectedTensorRules[R, Rdddd[a_, b_, c_, d_] /;
               OrderedQ[{a, b}] ∧ OrderedQ[{c, d}] ∧ OrderedQ[{{a, b}, {c, d}}]] //
           UseCoordinates[varnames] // Simplify) // TableForm
```

Out[20]//TableForm=

$$R_{t\,\theta\,t\,\theta} \to -r^2\,a[t]\,a''[t]$$
$$R_{t\,\phi\,t\,\phi} \to -r^2\,a[t]\,\mathrm{Sin}[\theta]^2\,a''[t]$$
$$R_{r\,t\,r\,t} \to \frac{a[t]\,a''[t]}{-1+k\,r^2}$$
$$R_{r\,\theta\,r\,\theta} \to -\frac{r^2\,a[t]^2\,(k+a'[t]^2)}{-1+k\,r^2}$$
$$R_{r\,\phi\,r\,\phi} \to -\frac{r^2\,a[t]^2\,\mathrm{Sin}[\theta]^2\,(k+a'[t]^2)}{-1+k\,r^2}$$
$$R_{\theta\,\phi\,\theta\,\phi} \to r^4\,a[t]^2\,\mathrm{Sin}[\theta]^2\,(k+a'[t]^2)$$

Hartle presents the results in terms of an orthonormal basis aligned with the coordinate basis. We first calculate the basis by using the inverse square roots of diagonal elements of the metric.

```
In[21]:= diagmetric{-1, 1, 1, 1}
         Simplify[%^-1/2] // PowerExpand;
         basismatrix = DiagonalMatrix[% // CoordinatesToTensors[varnames, x]];
         SetTensorValueRules[ed[red@μ], basismatrix]
         NonzeroValueRules[e] // UseCoordinates[varnames] // TableForm
```

$$\text{Out[21]=}\ \left\{1,\ \frac{a[t]^2}{1-k\,r^2},\ r^2\,a[t]^2,\ r^2\,a[t]^2\,\text{Sin}[\theta]^2\right\}$$

Out[25]//TableForm=

$$e_t \to \{1,\ 0,\ 0,\ 0\}$$
$$e_r \to \left\{0,\ \frac{\sqrt{1-k\,r^2}}{a[t]},\ 0,\ 0\right\}$$
$$e_\theta \to \left\{0,\ 0,\ \frac{1}{r\,a[t]},\ 0\right\}$$
$$e_\phi \to \left\{0,\ 0,\ 0,\ \frac{\text{Csc}[\theta]}{r\,a[t]}\right\}$$

We carry out the coordinate transformation by pre and post multiplying by Λ transformation matrices. We have to transpose the levels of the Riemann array when carrying out some of the matrix multiplications because *Mathematica* contracts the last level with the first level when multiplying arrays.

```
In[26]:= Λ = basismatrix;
         Transpose[Λ].riemannd.Λ;
         Transpose[Λ].Transpose[%, {2, 1, 4, 3}].Λ // Simplify;
         onriemannd = Transpose[%, {2, 1, 4, 3}];
         SetTensorValueRules[Rdddd[μ, ν, ρ, σ] // ToFlavor[red], onriemannd]
         (SelectedTensorRules[R, Rdddd[a_, b_, c_, d_] /; OrderedQ[{a, b}] ∧
                OrderedQ[{c, d}] ∧ OrderedQ[{{a, b}, {c, d}}] ∧ Head[a] === red] //
            UseCoordinates[varnames] // Simplify) // TableForm
```

Out[31]//TableForm=

$$R_{t\theta t\theta} \to -\frac{a''[t]}{a[t]}$$
$$R_{t\phi t\phi} \to -\frac{a''[t]}{a[t]}$$
$$R_{rtrt} \to -\frac{a''[t]}{a[t]}$$
$$R_{r\theta r\theta} \to \frac{k+a'[t]^2}{a[t]^2}$$
$$R_{r\phi r\phi} \to \frac{k+a'[t]^2}{a[t]^2}$$
$$R_{\theta\phi\theta\phi} \to \frac{k+a'[t]^2}{a[t]^2}$$

Restore the original state...

```
In[32]:= {gdd[a, b], guu[a, b], Γddd[a, b, c], Γudd[a, b, c],
           Rdddd[a, b, c, d], Rdddd[a, b, c, d] // ToFlavor[red]};
         ClearTensorValues /@ %;

In[34]:= ClearTensorShortcuts[{{x, e}, 1}, {{g, δ}, 2}, {Γ, 3}, {R, 4}]

In[35]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors, labs, varnames, metric, Λ, downΓ, upΓ, riemannd]
```

CalculateRRRG

- `CalculateRRRG[`*g, riemanndown, flavor : Identity, simplifyroutine : Identity*`]` will calculate Ruddd, Rdd, R and Gdd, the up version of the Riemann tensor, the Ricci tensor, the scalar curvature and the Einstein tensor.

g is the label for the metric matrix which must be defined.

The optional index flavor is the flavor used in the `g`. The optional argument simplifyroutine specifies the simplify routine to be applied to each independent element.

The tensors are returned as arrays in a list in the order: {riemann, ricci, scalar, einstein}.

See the note in `CalculateRiemannd` for conventions on the signs of curvature tensors.

See also: `CalculateRiemannd`, `SetTensorValues`, `SetRicciContraction`.

Example - Friedman-Robertson-Walker Geometry

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

Save the settings and declare base indices and flavors.

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

We extend the calculation for the Friedman-Robertson-Walker geometry from Appendix B in the Hartle *Gravitation* book.

```
In[5]:= varnames = {t, r, θ, ϕ};
        DeclareBaseIndices[varnames]
        DeclareIndexFlavor[{red, Red}]
        labs = {x, δ, g, Γ};
        DefineTensorShortcuts[{{x, e}, 1}, {{g, δ, R, G, L}, 2}, {Γ, 3}, {R, 4}]
```

The metric for proper distances is...

```
In[10]:= SetAttributes[k, Constant];
         diagmetric = {-1, a[t]^2 {1/(1 - k r^2), r^2 {1, Sin[θ]^2}}} // Flatten;
         (cmetric = DiagonalMatrix[diagmetric]) // MatrixForm
         metric = % // CoordinatesToTensors[varnames];
         MapThread[SetTensorValueRules[#1, #2] &,
           {{gdd[a, b], guu[a, b]}, {metric, Inverse[metric]}}];
```

```
Out[12]//MatrixForm=
⎛ -1   0          0            0                    ⎞
⎜                                                   ⎟
⎜  0   a[t]^2     0            0                    ⎟
⎜      ------                                       ⎟
⎜      1-k r^2                                      ⎟
⎜                                                   ⎟
⎜  0   0          r^2 a[t]^2   0                    ⎟
⎜                                                   ⎟
⎜  0   0          0            r^2 a[t]^2 Sin[θ]^2  ⎟
⎝                                                   ⎠
```

Next, calculate and set the Christoffel symbols.

```
In[15]:= {downΓ, upΓ} = CalculateChristoffels[labs];
         MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]}, {downΓ, upΓ}}];
         SelectedTensorRules[Γ, Γudd[_, j_, k_] /; OrderedQ[{j, k}]] //
           UseCoordinates[varnames] // TableForm
```

Out[17]//TableForm=

$$\Gamma^{t}{}_{r\,r} \to \frac{a[t]\,a'[t]}{1-k\,r^2}$$
$$\Gamma^{t}{}_{\theta\,\theta} \to r^2\,a[t]\,a'[t]$$
$$\Gamma^{t}{}_{\phi\,\phi} \to r^2\,a[t]\,\text{Sin}[\theta]^2\,a'[t]$$
$$\Gamma^{r}{}_{r\,t} \to \frac{a'[t]}{a[t]}$$
$$\Gamma^{r}{}_{r\,r} \to \frac{k\,r}{1-k\,r^2}$$
$$\Gamma^{r}{}_{\theta\,\theta} \to -r\,(1-k\,r^2)$$
$$\Gamma^{r}{}_{\phi\,\phi} \to -r\,(1-k\,r^2)\,\text{Sin}[\theta]^2$$
$$\Gamma^{\theta}{}_{t\,\theta} \to \frac{a'[t]}{a[t]}$$
$$\Gamma^{\theta}{}_{r\,\theta} \to \frac{1}{r}$$
$$\Gamma^{\theta}{}_{\phi\,\phi} \to -\text{Cos}[\theta]\,\text{Sin}[\theta]$$
$$\Gamma^{\phi}{}_{t\,\phi} \to \frac{a'[t]}{a[t]}$$
$$\Gamma^{\phi}{}_{r\,\phi} \to \frac{1}{r}$$
$$\Gamma^{\phi}{}_{\theta\,\phi} \to \text{Cot}[\theta]$$

We can then calculate the down form of the Riemann matrix.

```
In[18]:= riemannd = CalculateRiemannd[labs];
         SetTensorValueRules[Rdddd[μ, ν, σ, ρ], riemannd]
         (SelectedTensorRules[R, Rdddd[a_, b_, c_, d_] /;
               OrderedQ[{a, b}] ∧ OrderedQ[{c, d}] ∧ OrderedQ[{{a, b}, {c, d}}]] //
           UseCoordinates[varnames] // Simplify) // TableForm
```

Out[20]//TableForm=

$$R_{t\,\theta\,t\,\theta} \to r^2\,a[t]\,a''[t]$$
$$R_{t\,\phi\,t\,\phi} \to r^2\,a[t]\,\text{Sin}[\theta]^2\,a''[t]$$
$$R_{r\,t\,r\,t} \to \frac{a[t]\,a''[t]}{1-k\,r^2}$$
$$R_{r\,\theta\,r\,\theta} \to \frac{r^2\,a[t]^2\,(k+a'[t]^2)}{-1+k\,r^2}$$
$$R_{r\,\phi\,r\,\phi} \to \frac{r^2\,a[t]^2\,\text{Sin}[\theta]^2\,(k+a'[t]^2)}{-1+k\,r^2}$$
$$R_{\theta\,\phi\,\theta\,\phi} \to -r^4\,a[t]^2\,\text{Sin}[\theta]^2\,(k+a'[t]^2)$$

We calculate the other curvature tensors from the down Riemann array using CalculateRRRG. The calculation of the Ricci tensor depends upon which index of Riemann is used for contraction. The default is the first and third. It can be set to the first and fourth index by using...

```
In[21]:= SetRicciContraction[4]

Out[21]= 4

In[22]:= {riemann, ricci4, scalarcurvature, einstein} =
           CalculateRRRG[g, riemannd, Identity, Simplify[#, Trig → False] &];
```

But in this case we want to use the third index, which is the default setting. Recalculating...

```
In[23]:= SetRicciContraction[3]
         {riemann, ricci, scalarcurvature, einstein} =
           CalculateRRRG[g, riemannd, Identity, Simplify[#, Trig → False] &];

Out[23]= 3
```

The two Ricci tensors will have opposite signs.

```
In[25]:= ricci4 == -ricci // Simplify

Out[25]= True
```

Setting and looking at the up Riemann elements we obtain...

```
In[26]:= SetTensorValueRules[Ruddd[μ, ν, σ, ρ], riemann]
         (SelectedTensorRules[R, Ruddd[a_, b_, c_, d_] /; OrderedQ[{c, d}]] //
             UseCoordinates[varnames] // Simplify) // TableForm
```

Out[27]//TableForm=

$$R^{t}{}_{r\,r\,t} \to \frac{a[t]\,a''[t]}{1-k\,r^2}$$

$$R^{t}{}_{\theta\,t\,\theta} \to -r^2\,a[t]\,a''[t]$$

$$R^{t}{}_{\phi\,t\,\phi} \to -r^2\,a[t]\,\text{Sin}[\theta]^2\,a''[t]$$

$$R^{r}{}_{t\,r\,t} \to \frac{a''[t]}{a[t]}$$

$$R^{r}{}_{\theta\,r\,\theta} \to -r^2\,(k + a'[t]^2)$$

$$R^{r}{}_{\phi\,r\,\phi} \to -r^2\,\text{Sin}[\theta]^2\,(k + a'[t]^2)$$

$$R^{\theta}{}_{t\,t\,\theta} \to -\frac{a''[t]}{a[t]}$$

$$R^{\theta}{}_{r\,r\,\theta} \to \frac{k+a'[t]^2}{1-k\,r^2}$$

$$R^{\theta}{}_{\phi\,\theta\,\phi} \to -r^2\,\text{Sin}[\theta]^2\,(k + a'[t]^2)$$

$$R^{\phi}{}_{t\,t\,\phi} \to -\frac{a''[t]}{a[t]}$$

$$R^{\phi}{}_{r\,r\,\phi} \to \frac{k+a'[t]^2}{1-k\,r^2}$$

$$R^{\phi}{}_{\theta\,\theta\,\phi} \to r^2\,(k + a'[t]^2)$$

Setting and looking at the Ricci elements we obtain...

```
In[28]:= SetTensorValueRules[Rdd[μ, ν], ricci]
         (SelectedTensorRules[R, Rdd[a_, b_] /; OrderedQ[{a, b}]] //
             UseCoordinates[varnames] // Simplify) // TableForm
```

Out[29]//TableForm=

$$R_{t\,t} \to -\frac{3\,a''[t]}{a[t]}$$

$$R_{r\,r} \to \frac{2\,k+2\,a'[t]^2+a[t]\,a''[t]}{1-k\,r^2}$$

$$R_{\theta\,\theta} \to r^2\,(2\,(k + a'[t]^2) + a[t]\,a''[t])$$

$$R_{\phi\,\phi} \to r^2\,\text{Sin}[\theta]^2\,(2\,(k + a'[t]^2) + a[t]\,a''[t])$$

The scalar curvature is...

```
In[30]:= scalarcurvature // UseCoordinates[varnames]

Out[30]= 6 (k + a'[t]^2 + a[t] a''[t]) / a[t]^2
```

Setting and looking at the Einstein tensor elements we obtain...

```
In[31]:= SetTensorValueRules[Gdd[μ, ν], einstein]
         (SelectedTensorRules[G, Gdd[a_, b_] /; OrderedQ[{a, b}]] //
            UseCoordinates[varnames] // Simplify) // TableForm
```

Out[32]//TableForm=

$$G_{t\,t} \to \frac{3\,(k+a'[t]^2)}{a[t]^2}$$

$$G_{r\,r} \to \frac{k+a'[t]^2+2\,a[t]\,a''[t]}{-1+k\,r^2}$$

$$G_{\theta\,\theta} \to -r^2\,(k + a'[t]^2 + 2\,a[t]\,a''[t])$$

$$G_{\phi\,\phi} \to -r^2\,\text{Sin}[\theta]^2\,(k + a'[t]^2 + 2\,a[t]\,a''[t])$$

Hartle presents the results in terms of an orthonormal basis aligned with the coordinate basis. We use OrthonormalBasis to calculate the transformation matrix. We can ignore the warning messsage.

```
In[33]:= OrthonormalTransformation[cmetric,
           {-1, 1, 1, 1}, Simplify[#, r > 0 ∧ a[t] > 0 ∧ 0 < θ < π] &];
         Lmatrix = % // PowerExpand
         % // CoordinatesToTensors[varnames];
         SetTensorValues[Lud[a, red@α], %]
```

```
OrthonormalTransformation::signs :
 Warning, signs of eigenvalues {-1, 1/Sign[1 - k r^2], 1, 1} do not match signature
   {-1, 1, 1, 1} or are undetermined. Permutation matrix omitted for general case.
```

$$\text{Out[34]=}\ \left\{\{1, 0, 0, 0\}, \left\{0, \frac{\sqrt{1-k\,r^2}}{a[t]}, 0, 0\right\}, \left\{0, 0, \frac{1}{r\,a[t]}, 0\right\}, \left\{0, 0, 0, \frac{\text{Csc}[\theta]}{r\,a[t]}\right\}\right\}$$

The following gives the orthonormal basis in terms of the coordinate basis.

```
In[37]:= ed[red@α] == Lud[a, red@α] ed[a]
         % // ToArrayValues[] // PowerExpand // UseCoordinates[varnames]
```

$$\text{Out[37]=}\ e_\alpha == e_a\,L^a{}_\alpha$$

$$\text{Out[38]=}\ \left\{e_t == e_t,\ e_r == \frac{\sqrt{1-k\,r^2}\ e_r}{a[t]},\ e_\theta == \frac{e_\theta}{r\,a[t]},\ e_\phi == \frac{\text{Csc}[\theta]\ e_\phi}{r\,a[t]}\right\}$$

Calculating the orthonormal components of the Riemann tensor using array multiplication...

```
In[39]:= criemannd = riemannd // UseCoordinates[varnames];
         Lmatrix.Lmatrix.criemannd.Lmatrix.Lmatrix;
         SetTensorValueRules[Rdddd[μ, ν, ρ, σ] // ToFlavor[red], %]
         (SelectedTensorRules[R, Rdddd[a_, b_, c_, d_] /; Head[a] === red ∧
               OrderedQ[{a, b}] ∧ OrderedQ[{c, d}] ∧ OrderedQ[{{a, b}, {c, d}}]] //
            UseCoordinates[varnames] // Simplify) // TableForm
```

Out[42]//TableForm=

$$R_{t\,\theta\,t\,\theta} \to -\frac{a''[t]}{a[t]}$$

$$R_{t\,\phi\,t\,\phi} \to -\frac{a''[t]}{a[t]}$$

$$R_{r\,t\,r\,t} \to -\frac{a''[t]}{a[t]}$$

$$R_{r\,\theta\,r\,\theta} \to \frac{k+a'[t]^2}{a[t]^2}$$

$$R_{r\,\phi\,r\,\phi} \to \frac{k+a'[t]^2}{a[t]^2}$$

$$R_{\theta\,\phi\,\theta\,\phi} \to \frac{k+a'[t]^2}{a[t]^2}$$

Calculating the orthonormal components of the Ricci tensor by array multiplication...

```
In[43]:= onricci = Lmatrix.ricci.Lmatrix // Simplify;
         SetTensorValueRules[Rdd[μ, ν] // ToFlavor[red], onricci]
         (SelectedTensorRules[R, Rdd[a_, b_] /; OrderedQ[{a, b}] ∧ Head[a] === red] //
             UseCoordinates[varnames] // Simplify) // TableForm
```

Out[45]//TableForm=

$$R_{t\,t} \to -\frac{3\,a''[t]}{a[t]}$$

$$R_{r\,r} \to \frac{2\,k+2\,a'[t]^2+a[t]\,a''[t]}{a[t]^2}$$

$$R_{\theta\,\theta} \to \frac{2\,(k+a'[t]^2)+a[t]\,a''[t]}{a[t]^2}$$

$$R_{\phi\,\phi} \to \frac{2\,(k+a'[t]^2)+a[t]\,a''[t]}{a[t]^2}$$

Calculating the orthonormal components of the Einstein tensor by array multiplication...

```
In[46]:= oneinstein = Lmatrix.einstein.Lmatrix // Simplify;
         SetTensorValueRules[Gdd[μ, ν] // ToFlavor[red], oneinstein]
         (SelectedTensorRules[G, Gdd[a_, b_] /; OrderedQ[{a, b}] ∧ Head[a] === red] //
             UseCoordinates[varnames] // Simplify) // TableForm
```

Out[48]//TableForm=

$$G_{t\,t} \to \frac{3\,(k+a'[t]^2)}{a[t]^2}$$

$$G_{r\,r} \to -\frac{k+a'[t]^2+2\,a[t]\,a''[t]}{a[t]^2}$$

$$G_{\theta\,\theta} \to -\frac{k+a'[t]^2+2\,a[t]\,a''[t]}{a[t]^2}$$

$$G_{\phi\,\phi} \to -\frac{k+a'[t]^2+2\,a[t]\,a''[t]}{a[t]^2}$$

Restore the original state...

```
In[49]:=
         {gdd[i, j], guu[i, j], Rdddd[i, j, k, l], Rdddd @@ red /@ {i, j, k, l},
          Gdd @@ red /@ {i, j}, Rdd @@ red /@ {i, j}, Γddd[i, j, k], Γudd[i, j, k]}
         ClearTensorValues[% // Evaluate]
```

Out[49]= $\{g_{i\,j},\ g^{i\,j},\ R_{i\,j\,k\,l},\ R_{i\,j\,k\,l},\ G_{i\,j},\ R_{i\,j},\ \Gamma_{i\,j\,k},\ \Gamma^{i}{}_{j\,k}\}$

```
In[51]:= ClearTensorShortcuts[{{x, e}, 1}, {{g, δ, R, G, L}, 2}, {Γ, 3}, {R, 4}]
```

```
In[52]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
```

```
In[55]:= Clear[oldindices, oldflavors, labs, varnames, metric, Λ, downΓ, upΓ,
          Lmatrix, metric, cmetric, riemannd, criemannd, onricci, oneinstein]
```

CalculateWeyldown

- `CalculateWeyldown[`*metric, riemannd, ricci, Rscalar, simplifyroutine : Identity*`]` will calculate the all down Weyl conformal tensor..

riemannd is the all down riemann array. (Obtained with CalculateRiemannd)

ricci is the all down ricci array and Rscalar is the curvature scalar. (Obtained with CalculateRRRG)

The optional argument simplifyroutine specifies the simplify routine to be applied to the result.

The Weyl tensor can be nonzero only for a space of 4 or greater dimensions.

See the note in `CalculateRiemannd` for conventions on the signs of curvature tensors.

See also: `CalculateRiemannd`, `CalculateRRRG`.

Example - The Brinkmann Metric

The Brinkmann metric describes plane gravitational waves.

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

Save the settings and declare base indices and flavors.

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

```
In[3]:= DefineTensorShortcuts[{{x}, 1}, {{g, δ, R, zero}, 2}, {{Γ}, 3}, {{R, W}, 4}]
        DeclareZeroTensor[zero]
        labs = {x, δ, g, Γ};
```

Setting the base indices and metric.

```
In[6]:= vars = {u, v, x, y};
        DeclareBaseIndices[vars]
        simplifyroutine = Simplify;
        (cmetric = {{H[u, x, y], 1, 0, 0}, {1, 0, 0, 0}, {0, 0, -1, 0}, {0, 0, 0, -1}}) // MatrixForm
        (metric = cmetric // CoordinatesToTensors[]) // MatrixForm
        MapThread[SetTensorValues[#1, #2] &,
          {{gdd[a, b], guu[a, b]}, {metric, simplifyroutine@Inverse@metric}}];
```

Out[9]//MatrixForm=

$$\begin{pmatrix} H[u, x, y] & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Out[10]//MatrixForm=

$$\begin{pmatrix} H[x^u, x^x, x^y] & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Calculate the Christoffel symbols.

```
In[12]:= MapThread[SetTensorValues[#1, #2] &, {{Γddd[a, b, c], Γudd[a, b, c]},
            CalculateChristoffels[labs, Identity, simplifyroutine]}];
```

Calculate the Riemann tensor.

```
In[13]:= riemannd = CalculateRiemannd[labs, Identity, simplifyroutine];
```

Calculate the up Riemann, Ricci, Rscalar, and Einstein tensors.

```
In[14]:= {riemannu, ricci, Rscalar, einstein} =
           CalculateRRRG[g, riemannd, Identity, simplifyroutine];
```

Calculate the Weyl tensor.

```
In[15]:= weyldown = CalculateWeyldown[metric, riemannd, ricci, Rscalar, simplifyroutine];
```

```
In[16]:= SetTensorValueRules[Wdddd[a, b, c, d], weyldown];
         NonzeroValueRules[W] // UseCoordinates[] // TableForm
```

Out[17]//TableForm=

$$W_{uxux} \to \frac{1}{4}\left(H^{(0,0,2)}[u, x, y] - H^{(0,2,0)}[u, x, y]\right)$$
$$W_{uxuy} \to -\frac{1}{2} H^{(0,1,1)}[u, x, y]$$
$$W_{uxxu} \to \frac{1}{4}\left(-H^{(0,0,2)}[u, x, y] + H^{(0,2,0)}[u, x, y]\right)$$
$$W_{uxyu} \to \frac{1}{2} H^{(0,1,1)}[u, x, y]$$
$$W_{uyux} \to -\frac{1}{2} H^{(0,1,1)}[u, x, y]$$
$$W_{uyuy} \to \frac{1}{4}\left(-H^{(0,0,2)}[u, x, y] + H^{(0,2,0)}[u, x, y]\right)$$
$$W_{uyxu} \to \frac{1}{2} H^{(0,1,1)}[u, x, y]$$
$$W_{uyyu} \to \frac{1}{4}\left(H^{(0,0,2)}[u, x, y] - H^{(0,2,0)}[u, x, y]\right)$$
$$W_{xuux} \to \frac{1}{4}\left(-H^{(0,0,2)}[u, x, y] + H^{(0,2,0)}[u, x, y]\right)$$
$$W_{xuuy} \to \frac{1}{2} H^{(0,1,1)}[u, x, y]$$
$$W_{xuxu} \to \frac{1}{4}\left(H^{(0,0,2)}[u, x, y] - H^{(0,2,0)}[u, x, y]\right)$$
$$W_{xuyu} \to -\frac{1}{2} H^{(0,1,1)}[u, x, y]$$
$$W_{yuux} \to \frac{1}{2} H^{(0,1,1)}[u, x, y]$$
$$W_{yuuy} \to \frac{1}{4}\left(H^{(0,0,2)}[u, x, y] - H^{(0,2,0)}[u, x, y]\right)$$
$$W_{yuxu} \to -\frac{1}{2} H^{(0,1,1)}[u, x, y]$$
$$W_{yuyu} \to \frac{1}{4}\left(-H^{(0,0,2)}[u, x, y] + H^{(0,2,0)}[u, x, y]\right)$$

An identity for the Weyl tensor is that the contraction on the 2nd and 4th slots is zero.

```
In[18]:= guu[b, d] Wdddd[a, d, c, b] == zerodd[a, c]
         % // MetricSimplify[g]
         (ToArrayValues[] /@ %%) // Simplify
```

Out[18]= $g^{bd} W_{adcb} == zero_{ac}$

Out[19]= $W_{adc}{}^{d} == zero_{ac}$

Out[20]= True

```
In[21]:=
         {gdd[i, j], guu[i, j], Γddd[i, j, k], Γudd[i, j, k], Wdddd[a, b, c, d]}
         ClearTensorValues[% // Evaluate]
```

Out[21]= $\{g_{ij}, g^{ij}, \Gamma_{ijk}, \Gamma^{i}{}_{jk}, W_{abcd}\}$

```
In[23]:= ClearTensorShortcuts[{{x}, 1}, {{g, δ, R, zero}, 2}, {{Γ}, 3}, {{R, W}, 4}]

In[24]:= DeclareBaseIndices @@ oldindices

In[25]:= Clear[oldindices, labs, vars, metric,
          cmetric, riemannd, riemannu, ricci, Rscalar, einstein]
```

CanonicalUpDowns

- `CanonicalUpDowns[`*term*`]` will perform UpDownSwaps on dummy indices to put the ups in the lowest *Mathematica* position.

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

See also: `UpDownAdjust`.

Examples

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

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

Some examples of using `CanonicalUpDowns`...

```
In[4]:= testcase = {pd[a] Tud[a, b],
           2 q Rdd[a, b] Ruudd[a, b, c, d], pd[a] PartialD[labs99][pu[a], xu[b]]};
        Thread[testcase → CanonicalUpDowns /@ testcase] // TableForm
```

```
Out[5]//TableForm=
```

$$p_a\ T^a{}_b \to p^a\ T_{a\,b}$$

$$2\ q\ R_{a\,b}\ R^{a\,b}{}_{c\,d} \to 2\ q\ R^{a\,b}\ R_{a\,b\,c\,d}$$

$$p_a\ \frac{\partial p^a}{\partial x^b} \to p^a\ \frac{\partial p_a}{\partial x^b}$$

```
In[6]:= ClearTensorShortcuts[{{x, p}, 1}, {{R, T}, 2}, {R, 4}]

In[7]:= Clear[testcase, labs99]
```

CircleEvalRule

■ `CircleEvalRule` evaluates direct products using CircleTimes on sets of arguments.

CircleEvalRule is used by EvaluateSlots

It performs transformations such as `(v1⊗v2⊗v3⊗v4)[w1, , w3,]⟹v2⊗v4 v1.w1 v3.w3`.

The CircleTimes expression may be replaced by a single Tensor, but it can't be replaced by a single symbol because then the rule might apply where it is not wanted. Instead, use `CircleTimes[symbol]`.

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

Examples

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

The following is the simplest example. Here the head is a tensor.

```
In[3]:= gd[i][gu[j]]
        % /. CircleEvalRule
```

$$\text{Out[3]= } g_i\left[g^j\right]$$

$$\text{Out[4]= } g_i.g^j$$

In the following the round parentheses are necessary. This is a complete evaluation to a scalar.

```
In[5]:= (u⊗v⊗w)[x, y, z]
        % /. CircleEvalRule
```

```
Out[5]= (u⊗v⊗w)[x, y, z]
```

```
Out[6]= u.x v.y w.z
```

Null slots are not evaluated so the following returns a second order tensor.

```
In[7]:= (u⊗v⊗w)[, , z]
        % /. CircleEvalRule
```

```
Out[7]= (u⊗v⊗w)[Null, Null, z]
```

```
Out[8]= u⊗v w.z
```

The following will not evaluate because it is too common an expression.

```
In[9]:= v[z]
        % /. CircleEvalRule
```

```
Out[9]= v[z]
```

```
Out[10]= v[z]
```

It can instead be entered this way...

```
In[11]:= CircleTimes[v][z]
         % /. CircleEvalRule

Out[11]= (⨂v)[z]

Out[12]= v.z
```

CircleEvalRule does not do any further evaluation. EvaluateSlots fully evaluates.

```
In[13]:= (u⊗v)[x, y]
         % /. {u → ud[i] gu[i], v → vd[i] gu[i], x → xu[j] gd[j], y → yu[j] gd[j]}
         % /. CircleEvalRule

Out[13]= (u⊗v)[x, y]

Out[14]= ((gⁱ uᵢ)⊗(gⁱ vᵢ))[gⱼ xʲ, gⱼ yʲ]

Out[15]= (gⁱ uᵢ).(gⱼ xʲ) (gⁱ vᵢ).(gⱼ yʲ)

In[16]:= (u⊗v)[x, y]
         % /. {u → ud[i] gu[i], v → vd[j] gu[j], x → xu[m] gd[m], y → yu[n] gd[n]}
         % // EvaluateSlots[g, δ]

Out[16]= (u⊗v)[x, y]

Out[17]= ((gⁱ uᵢ)⊗(gʲ vⱼ))[gₘ xᵐ, gₙ yⁿ]

Out[18]= uₘ vₙ xᵐ yⁿ

In[19]:= ClearTensorShortcuts[{u, v, x, y, g}, 1]
```

CircleTimes

■ `CircleTimes` is used to form tensor products.

`Times` can't be used for tensor products because it is unordered.

`CircleTimes` can be entered from the BasicTypesetting palette, or from the CompleteCharacters palette under Operators\-General. It can also be entered as esc c* esc, or as \[CircleTimes] without the space.

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

Examples

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

Let `u` and `v` be two vectors. Then a tensor dyad is formed by...

```
In[2]:= u⊗v

Out[2]= u⊗v
```

Tensor products can be evaluated on a list of arguments

```
In[3]:= (u⊗v⊗w)[x, y, z]
        % /. CircleEvalRule

Out[3]= (u⊗v⊗w)[x, y, z]

Out[4]= u.x v.y w.z
```

ClearIndexFlavor

- `ClearIndexFlavor[` *flavorname* ... `]` will delete the index flavors from the IndexFlavors list and clear the associated Format statements.
- `ClearIndexFlavor[{` *flavorname, flavorform*} ... `]` can also be used.
- `ClearIndexFlavor[{{` *flavorname, flavorform*} ...}`]` can also be used to clear the existing `IndexFlavors`.

See `ToFlavor` for examples of using flavored indices.

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

Examples

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

The following gives the current set of of index flavors.

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

This declares new index flavors.

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

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

Flavored indices are then displayed with the designated forms.

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

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

This clears the index flavors...

```
In[7]:= ClearIndexFlavor[IndexFlavors];
        IndexFlavors

Out[8]= {}
```

The flavors are no longer recognized.

```
In[9]:= {Tensor[x, {red[i]}, {Void}], Tensor[x, {rocket[i]}, {Void}]}

Out[9]= {x^red[i], x^rocket[i]}
```

This resets to the original flavors.

```
In[10]:= DeclareIndexFlavor /@ oldflavors;
         Clear[oldflavors]
```

ClearTensorShortcuts

- `ClearTensorShortcuts[`*label, order*`]` will clear all tensor shortcuts established by `DefineTensorShortcute[label, order]`.
- `ClearTensorShortcuts[{`*u, v, w, ...*`}, `*order*`]` will clear a series of tensor shortcuts of the same order.
- `ClearTensorShortcuts[{{`*u, v, w, ...*`}, `*order*`}..]` will clear a number of tensors of different orders.

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

Examples

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

This defines all the shortcuts for a second order tensor R.

```
In[2]:= DefineTensorShortcuts[R, 2]
```

```
In[3]:= #[i, j] & /@ {Ruu, Rud, Rdu, Rdd}
```

```
Out[3]= {R^{i j}, R^{i}_{j}, R_{i}^{j}, R_{i j}}
```

The definitions are stored under the shortcut names.

```
In[4]:= Information /@ {Ruu, Rud, Rdu, Rdd};
          Global`Ruu
          Ruu[u1_, u2_] := R^{u1 u2}
          Global`Rud
          Rud[u3_, d1_] := R^{u3}_{d1}
          Global`Rdu
          Rdu[d2_, u4_] := R_{d2}^{u4}
          Global`Rdd
          Rdd[d3_, d4_] := R_{d3 d4}
```

This clears the definitions.

```
In[5]:= ClearTensorShortcuts[R, 2]
```

```
In[6]:= #[i, j] & /@ {Ruu, Rud, Rdu, Rdd}
```

```
Out[6]= {Ruu[i, j], Rud[i, j], Rdu[i, j], Rdd[i, j]}
```

```
In[7]:= Information /@ {Ruu, Rud, Rdu, Rdd};
          Global`Ruu
          Global`Rud
          Global`Rdu
          Global`Rdd
```

ClearTensorValues

- `ClearTensorValues[`*Tensor*[*label, upindices, downindices*]] will clear any component value definitions of the tensor from `UpValues[label]` and remove any substitution rules from `TensorValueRules[label]`.
- `ClearTensorValues[`*Tensor*[*label*]] will clear any values and rules for a scalar tensor.
- `ClearTensorValues[{`*tensor1, tensor2* ...}] will clear a number of tensor values or rules.

Definitions or rules that are based on the same label but with a different index forms or structures will not be removed.

The tensors may be entered in shortcut form.

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

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[T, 1]
```

This creates tensor values in the red frame and a value for a scalar tensor ϕ.

```
In[8]:= Clear[T, ϕ];
        SetTensorValues[Tu[i] // ToFlavor[red], {1, 2, 3}]
        SetTensorValues[Tensor[ϕ], t^2]
        UpValues[T]
        UpValues[ϕ]

Out[11]= {HoldPattern[T^1] :↦ 1, HoldPattern[T^2] :↦ 2, HoldPattern[T^3] :↦ 3}

Out[12]= {HoldPattern[ϕ] :↦ t^2}
```

This creates values for `T` in the plain frame. (We have simply picked arbitrary values for the components.) The plain definitions have been added to the red definitions.

```
In[13]:= SetTensorValues[Tu[i], {4, 5, 6}]
         UpValues[T]

Out[14]= {HoldPattern[T^1] :↦ 4, HoldPattern[T^2] :↦ 5, HoldPattern[T^3] :↦ 6,
           HoldPattern[T^1] :↦ 1, HoldPattern[T^2] :↦ 2, HoldPattern[T^3] :↦ 3}
```

Now, we clear the red definitions...

```
In[15]:= ClearTensorValues[Tu[i] // ToFlavor[red]]
         UpValues[T]

Out[16]= {HoldPattern[T^1] :> 4, HoldPattern[T^2] :> 5, HoldPattern[T^3] :> 6}
```

...and then the plain definitions...

```
In[17]:= ClearTensorValues[Tu[i]]
         UpValues[T]

Out[18]= {}
```

...and the scalar tensor value.

```
In[19]:= ClearTensorValues[Tensor[ϕ]]
         UpValues[ϕ]

Out[20]= {}
```

We can follow the same procedure with value substitution rules. This creates all the substitution rules.

```
In[21]:= SetTensorValueRules[Tu[i], {4, 5, 6}]
         SetTensorValueRules[Tu[i] // ToFlavor[red], {1, 2, 3}]
         SetTensorValueRules[Tensor[ϕ], t^2]
         TensorValueRules[T, ϕ]

Out[24]= {T^1 → 4, T^2 → 5, T^3 → 6, T^1 → 1, T^2 → 2, T^3 → 3, ϕ → t^2}
```

Now, we clear the red rules...

```
In[25]:= ClearTensorValues[Tu[i] // ToFlavor[red]]
         TensorValueRules[T, ϕ]

Out[26]= {T^1 → 4, T^2 → 5, T^3 → 6, ϕ → t^2}
```

...and the plain rules...

```
In[27]:= ClearTensorValues[Tu[i]]
         TensorValueRules[T, ϕ]

Out[28]= {ϕ → t^2}
```

...and the scalar tensor.

```
In[29]:= ClearTensorValues[Tensor[ϕ]]
         TensorValueRules[T, ϕ]

Out[30]= {}
```

Set them all again.

```
In[31]:= SetTensorValueRules[Tu[i], {4, 5, 6}]
         SetTensorValueRules[Tu[i] // ToFlavor[red], {1, 2, 3}]
         SetTensorValueRules[Tensor[ϕ], t^2]
         TensorValueRules[T, ϕ]

Out[34]= {T^1 → 4, T^2 → 5, T^3 → 6, T^1 → 1, T^2 → 2, T^3 → 3, ϕ → t^2}
```

This clears them all at once.

```
In[35]:= ClearTensorValues[{Tensor[ϕ], Tu[i], Tu[red@i]}]
         TensorValueRules[T, ϕ]

Out[36]= {}

In[37]:= ClearTensorShortcuts[T, 1]
```

Restore the original state...

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

CompleteBaseIndices

- `CompleteBaseIndices` gives the current set of all base indices including those associated with various flavors.

CompleteBaseIndices are set by `DeclareBaseIndices`.

If you save `oldindicies = CompleteBaseIndices`, then you can reestablish the base indices by `DeclareBaseIndices @@ oldindices`.

On loading, Tensorial automatically initializes the base indices to `{1, 2, 3}` and the `CompleteBaseIndices` are `{{1, 2, 3}}`.

Base indices are not in themselves flavored, but flavors may be added to them.

See also: `BaseIndices`, `DeclareBaseIndices`, `NDim`, `BaseIndexQ`.

Examples

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

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

The following gives the current set of base indices...

```
In[4]:= oldindices = CompleteBaseIndices

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

The base indices can be changed with

```
In[5]:= DeclareBaseIndices[{t, x, y, z}, {red, {A, B}}]
        CompleteBaseIndices

Out[6]= {{t, x, y, z}, {red, {A, B}}}
```

This resets to the old indices.

```
In[7]:= DeclareBaseIndices @@ oldindices;
        DeclareIndexFlavor @@ oldflavors;
        CompleteBaseIndices
        Clear[oldindices, oldflavors]

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

ConstructDirectProduct

- `ConstructDirectProduct[`*expr*`]` will convert a direct product of tensor slot expressions to an expression with a single set of slots.

The direct product is constructed using `CircleTimes`.

See also: `EvaluateSlots`, `CircleEvalRule`.

Examples

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

The following is a simple construction of a direct product.

```
In[2]:= S[w, x] ⊗ T[y, z]
        % // ConstructDirectProduct

Out[2]= S[w, x] ⊗ T[y, z]

Out[3]= (S ⊗ T)[w, x, y, z]
```

In the following, the direct product is constructed from two dyads and then evaluated.

```
In[4]:= (a ⊗ b)[w, x] ⊗ (c ⊗ d)[y, z]
        % // ConstructDirectProduct
        % /. CircleEvalRule

Out[4]= (a ⊗ b)[w, x] ⊗ (c ⊗ d)[y, z]

Out[5]= (a ⊗ b ⊗ c ⊗ d)[w, x, y, z]

Out[6]= a.w b.x c.y d.z

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

The following is constructed from two 1-forms on vectors and then evaluated.

```
In[8]:= (ud[m] eu[m])[xu[i] ed[i]] ⊗ (vd[n] eu[n])[yu[j] ed[j]]
        % // ConstructDirectProduct
        % // EvaluateSlots[e, g]
        % // EinsteinSum[] // Factor

Out[8]= (e^m u_m)[e_i x^i] ⊗ (e^n v_n)[e_j y^j]

Out[9]= ((e^m u_m) ⊗ (e^n v_n))[e_i x^i, e_j y^j]

Out[10]= u_i v_j x^i y^j

Out[11]= (u_1 x^1 + u_2 x^2 + u_3 x^3) (v_1 y^1 + v_2 y^2 + v_3 y^3)
```

This does the expansion before the direct product is constructed.

```
In[12]:= (ud[m] eu[m])[xu[i] ed[i]] ⊗ (vd[n] eu[n])[yu[j] ed[j]]
         % /. a_ b_ :→ EinsteinSum[][a b]
         % // ConstructDirectProduct
         % // EvaluateSlots[e, g] // Factor
```

$$\text{Out[12]= } (e^m u_m)\,[e_i\, x^i] \otimes (e^n v_n)\left[e_j\, y^j\right]$$

$$\text{Out[13]= } (e^1 u_1 + e^2 u_2 + e^3 u_3)\,[e_1 x^1 + e_2 x^2 + e_3 x^3] \otimes (e^1 v_1 + e^2 v_2 + e^3 v_3)\,[e_1 y^1 + e_2 y^2 + e_3 y^3]$$

$$\begin{aligned}\text{Out[14]= } &((e^1 u_1) \otimes (e^1 v_1) + (e^1 u_1) \otimes (e^2 v_2) + (e^1 u_1) \otimes (e^3 v_3) + (e^2 u_2) \otimes (e^1 v_1) + (e^2 u_2) \otimes (e^2 v_2) + \\ &\quad (e^2 u_2) \otimes (e^3 v_3) + (e^3 u_3) \otimes (e^1 v_1) + (e^3 u_3) \otimes (e^2 v_2) + (e^3 u_3) \otimes (e^3 v_3))\,[\\ &\ e_1 x^1 + e_2 x^2 + e_3 x^3,\ e_1 y^1 + e_2 y^2 + e_3 y^3]\end{aligned}$$

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

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

ContractArray

- `ContractArray[`*tarray, {lev1, lev2}...*`]` will contract the indicated pairs of level numbers in the tensor array tarray

The purpose of ContractArray is to perform contractions on the full matrix or array forms of tensors. It can be used in conjunction with the *dot mode* commands in the Arrays Section. Generally it is easier to use tensor index mode methods.

When tensors are expanded to arrays in Tensorial, the levels are always determined by the sort order of the raw indices, with the lowest sort order index at the highest level.

See also: `DotTensorFactors`, `ExpandDotArray`, `DotOperate`, `SumExpansion`, `ToArrayValues`, `Array-Expansion`, `EinsteinSum`.

Examples

See also the second Example, Arrays & Tensors.

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

Save the old settings.

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

```
In[4]:= DefineTensorShortcuts[T, 3]
```

The normal way to contract a tensor is to repeat an index in an up and down position. The following contracts the first and third slots of the tensor T.

```
In[5]:= Tuud[i, j, i]
        % // EinsteinSum[]
        % // EinsteinArray[] // MatrixForm
```

$$\text{Out[5]=}\quad T^{i\,j}{}_{i}$$

$$\text{Out[6]=}\quad T^{1\,j}{}_{1} + T^{2\,j}{}_{2} + T^{3\,j}{}_{3}$$

$$\text{Out[7]//MatrixForm=}\quad \begin{pmatrix} T^{1\,1}{}_{1} + T^{2\,1}{}_{2} + T^{3\,1}{}_{3} \\ T^{1\,2}{}_{1} + T^{2\,2}{}_{2} + T^{3\,2}{}_{3} \\ T^{1\,3}{}_{1} + T^{2\,3}{}_{2} + T^{3\,3}{}_{3} \end{pmatrix}$$

Sometimes, you may wish to contract an array after it has been expanded to a *Mathematica* array. ContractArray can then be used. The following contracts the first and third levels of T.

```
In[8]:= Tuud[i, j, k]
        % // EinsteinArray[]
        ContractArray[%, {1, 3}] // MatrixForm
```

```
Out[8]= T^{i j}_{k}
```

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

```
Out[10]//MatrixForm=
        ⎛ T^{1 1}_{1} + T^{2 1}_{2} + T^{3 1}_{3} ⎞
        ⎜ T^{1 2}_{1} + T^{2 2}_{2} + T^{3 2}_{3} ⎟
        ⎝ T^{1 3}_{1} + T^{2 3}_{2} + T^{3 3}_{3} ⎠
```

Restore settings.

```
In[11]:= ClearTensorShortcuts[T, 3]
```

```
In[12]:= DeclareBaseIndices @@ oldindices
         Clear[oldindices]
```

CoordinatesToTensors

- `CoordinateToTensors[{r, θ, ϕ ...}, coord : x, flavor : Identity][expr]` will convert the coordinate symbols in the expression to the corresponding indexed coordinate positions.

- `CoordinateToTensors[coord : x, flavor : Identity][expr]` uses the base indices associated with flavor as symbols, provided they are symbols.

The optional arguments coord and flavor give the coordinate label and index flavor to use. Their default values are x and Identity (plain).

The number of symbols must match the length of the base index set associated with the flavor of the index.

The first symbol will be converted to a tensor with the first base index and the last symbol will be converted to a tensor with the last base index.

See also: `UseCoordinates`, `DeclareBaseIndices`, `SetMetricValues`, `SetChristoffelValues`.

Examples

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

Save settings

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

In[7]:= DefineTensorShortcuts[{{x, y}, 1}, {{g}, 2}, {Γ, 3}]
```

`CoordinatesToTensors` is the inverse of `UseCoordinates`.

```
In[8]:= xu[i]
        % // EinsteinArray[]
        % // UseCoordinates[{x, y, z}]
        % // CoordinatesToTensors[{x, y, z}]

Out[8]= x^i

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

Out[10]= {x, y, z}

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

For flavored indices, and for coordinate symbols other than x, we have to include the optional arguments.

```
In[12]:= yu[i] // ToFlavor[red]
         % // EinsteinArray[]
         % // UseCoordinates[{x, y, z}, y, red]
         % // CoordinatesToTensors[{x, y, z}, y, red]
```

Out[12]= y^i

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

Out[14]= {x, y, z}

Out[15]= $\{y^1, y^2, y^3\}$

CoordinatesToTensors is useful when setting the metric tensor, which must be in terms of indexed coordinate positions. Let's set the Schwarzschild spacetime metric. We give symbolic values for the base indices.

```
In[16]:= SetAttributes[M, Constant]
         DeclareBaseIndices[{t, r, θ, ϕ}]
         cmetric = DiagonalMatrix[{-(1 - 2 M / r), 1 / (1 - 2 M / r), r^2, r^2 Sin[θ]^2}]
```

Out[18]= $\left\{\left\{-1+\frac{2M}{r}, 0, 0, 0\right\}, \left\{0, \frac{1}{1-\frac{2M}{r}}, 0, 0\right\}, \{0, 0, r^2, 0\}, \{0, 0, 0, r^2 \text{Sin}[\theta]^2\}\right\}$

We can then convert to the form required for calculating Christoffel symbols. Since we have used symbolic base indices we can use the argumentless form of CoordinatesToTensors.

```
In[19]:= (metric = cmetric // CoordinatesToTensors[]) // MatrixForm
         MapThread[SetTensorValueRules[#1, #2] &,
           {{gdd[a, b], guu[a, b]}, {metric, Inverse[metric]}}];
```

Out[19]//MatrixForm=

$$\begin{pmatrix} -1+\frac{2M}{x^r} & 0 & 0 & 0 \\ 0 & \frac{1}{1-\frac{2M}{x^r}} & 0 & 0 \\ 0 & 0 & (x^r)^2 & 0 \\ 0 & 0 & 0 & \text{Sin}[x^\theta]^2 (x^r)^2 \end{pmatrix}$$

We can then calculate the Christoffel symbols and use SelectedTensorRules and UseCoordinates to get to get the independent up symbols in coordinates symbol form.

```
In[21]:= {Γup, Γdown} = CalculateChristoffels[{x, δ, g, Γ}, Identity, Simplify];
         SetTensorValueRules[Γudd[a, b, c], Γup]
         SelectedTensorRules[Γ, Γudd[_, a_, b_] /; OrderedQ[{a, b}]] // UseCoordinates[] //
          TableForm
```

Out[23]//TableForm=

$\Gamma^t{}_{r\,t} \to -\frac{M}{r^2}$

$\Gamma^r{}_{t\,t} \to \frac{M}{r^2}$

$\Gamma^r{}_{r\,r} \to -\frac{M}{(-2M+r)^2}$

$\Gamma^r{}_{\theta\,\theta} \to -r$

$\Gamma^r{}_{\phi\,\phi} \to -r\,\text{Sin}[\theta]^2$

$\Gamma^\theta{}_{r\,\theta} \to r$

$\Gamma^\theta{}_{\phi\,\phi} \to -\frac{1}{2} r^2 \text{Sin}[2\theta]$

$\Gamma^\phi{}_{r\,\phi} \to r\,\text{Sin}[\theta]^2$

$\Gamma^\phi{}_{\theta\,\phi} \to \frac{1}{2} r^2 \text{Sin}[2\theta]$

`CoordinatesToTensors` can be used with multiple base indices.

```
In[24]:= DeclareBaseIndices[{x, y, z}, {red, {ρ, θ, ϕ}}]

In[25]:= {x, y, z}
         % // CoordinatesToTensors[]

Out[25]= {x, y, z}

Out[26]= {x^x, x^y, x^z}

In[27]:= {ρ, θ, ϕ}
         % // CoordinatesToTensors[x, red]

Out[27]= {ρ, θ, ϕ}

Out[28]= {x^ρ, x^θ, x^ϕ}
```

Restore settings

```
In[29]:= ClearTensorShortcuts[{{x, y}, 1}, {{g}, 2}]

In[30]:= ClearTensorValues[{Γudd[i, j, k], Γddd[i, j, k]}]

In[31]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors]

In[35]:= Clear[oldindices, oldflavors, Γup, Γdown]
```

CovariantCommutator

- `CovarianatCommutator[{c1, c2}, R, d][term]` will calculate the covariant commutator, CovariantD[term, {c1,c2}] - CovariantD[term,{c2,c1}] and express the result in terms of the Riemann tensor R. d is the dummy index that is introduced.

R is the label to be used for the Riemann tensor.

For flavored expressions the flavor must be on c1, c2 and d.

See also: `CalculateRiemannd`.

Example

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

```
In[2]:= oldflavors = IndexFlavors;
        DefineTensorShortcuts[{λ, 1}, {τ, 2}, {τ, 3}]
        DeclareIndexFlavor[{red, Red}]
```

A down index gives a positive term.

```
In[5]:= λd[a]
        CovariantD[Tensor[%], {b, c}] - CovariantD[Tensor[%], {c, b}] ==
         CovariantCommutator[{b, c}, R, d][%]
```

```
Out[5]= λ_a
```

```
Out[6]= (λ_a)_;bc - (λ_a)_;cb == R^d_abc λ_d
```

An up index gives a negative term.

```
In[7]:= λu[a]
        CovariantD[Tensor[%], {b, c}] - CovariantD[Tensor[%], {c, b}] ==
         CovariantCommutator[{b, c}, R, d][%]
```

```
Out[7]= λ^a
```

```
Out[8]= (λ^a)_;bc - (λ^a)_;cb == -R^a_dbc λ^d
```

There is a term for each index.

```
In[9]:= τuu[a, b]
        CovariantD[Tensor[%], {b, c}] - CovariantD[Tensor[%], {c, b}] ==
         CovariantCommutator[{c, d}, R, e][%]
```

```
Out[9]= τ^ab
```

```
Out[10]= (τ^ab)_;bc - (τ^ab)_;cb == -R^b_ecd τ^ae - R^a_ecd τ^eb
```

A term can contain products of tensors.

```
In[11]:= λu[a] λd[b]
         CovariantD[Tensor[%], {b, c}] - CovariantD[Tensor[%], {c, b}] ==
          CovariantCommutator[{c, d}, R, e][%]
```

Out[11]= $\lambda^a \lambda_b$

Out[12]= $(\lambda^a \lambda_b)_{;bc} - (\lambda^a \lambda_b)_{;cb} == -R^a{}_{ecd} \lambda^e \lambda_b + R^e{}_{bcd} \lambda^a \lambda_e$

```
In[13]:= τuud[a, b, c]
         CovariantD[%, {d, e}] - CovariantD[%, {e, d}] == CovariantCommutator[{d, e}, R, f][%]
```

Out[13]= $\tau^{ab}{}_c$

Out[14]= $\tau^{ab}{}_{c;de} - \tau^{ab}{}_{c;ed} == R^f{}_{cde} \tau^{ab}{}_f - R^b{}_{fde} \tau^{af}{}_c - R^a{}_{fde} \tau^{fb}{}_c$

The expansion is only on up and down indices.

```
In[15]:= λu[a] λd[a]
         CovariantD[Tensor[%], {b, c}] - CovariantD[Tensor[%], {c, b}] ==
          CovariantCommutator[{b, c}, R, d][%]
```

Out[15]= $\lambda^a \lambda_a$

Out[16]= $(\lambda^a \lambda_a)_{;bc} - (\lambda^a \lambda_a)_{;cb} == 0$

```
In[17]:= λu[a] τud[b, a]
         CovariantD[Tensor[%], {b, c}] - CovariantD[Tensor[%], {c, b}] ==
          CovariantCommutator[{c, d}, R, e][%]
```

Out[17]= $\lambda^a \tau^b{}_a$

Out[18]= $(\lambda^a \tau^b{}_a)_{;bc} - (\lambda^a \tau^b{}_a)_{;cb} == -R^b{}_{ecd} \lambda^a \tau^e{}_a$

For flavored expressions the flavor must be on the covariant and dummy indices.

```
In[19]:= λu[a] τud[b, c] // ToFlavor[red]
         CovariantD[Tensor[%], {red@b, red@c}] - CovariantD[Tensor[%], {red@c, red@b}] ==
          CovariantCommutator[{red@c, red@d}, R, red@e][%]
```

Out[19]= $\lambda^a \tau^b{}_c$

Out[20]= $(\lambda^a \tau^b{}_c)_{;bc} - (\lambda^a \tau^b{}_c)_{;cb} == -R^a{}_{ecd} \lambda^e \tau^b{}_c + R^e{}_{ccd} \lambda^a \tau^b{}_e - R^b{}_{ecd} \lambda^a \tau^e{}_c$

A term with bad indices will abort the operation.

```
In[21]:= λu[a] τud[a, a]
         % // CovariantCommutator[{b, c}, R, d]
```

Out[21]= $\lambda^a \tau^a{}_a$

```
CovariantCommutator::indices : The term λ^a τ^a_a has bad indices {a}.
```

```
Out[22]= $Aborted
```

Restore state.

```
In[23]:= ClearTensorShortcuts[{λ, 1}, {τ, 2}, {τ, 3}]
```

```
In[24]:= DeclareIndexFlavor @@ oldflavors;
```

```
In[25]:= Clear[oldflavors]
```

CovariantD

- `CovariantD[`*expr, i*`]` represents the covariant derivative of the tensor expression with respect to the index `i`.
- `CovariantD[`*expr,* {*i, j, ...*}`]` represents the covariant derivative of the expression with respect to the list of indices.

Covariant derivatives undergo linear and Liebnizian expansion but are otherwise inert until ExpandCovariantD is used..

With multiple differentiation indices, the differentiations are done left to right in accordance with the usage in most texts. This convention change was made to Tensorial in Nov 2006.

Covariant derivatives may only be expanded on tensors. They will not be expanded on expressions that contain base indices or partial derivatives. Use `ExpandCovariantD` on a symbolic tensor expression and then expand to base indices.

By default the covariant derivatives are prefixed by a single semicolon in the output display. The format can be changed using `SetDerivativeSymbols`. The display can also be changed to a subscripted ∇ symbol using SetCovariantDisplay.

The intended flavors must be on the indices in `CovariantD`.

See also: `ExpandCovariantD`, `SetScalarSingleCovariantD`, `PartialD`, `AbsoluteD`, `TotalD`, `SetDerivativeSymbols`, `SetCovariantDisplay`.

Examples

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

Save the settings.

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

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

Here is a tensor T and its covariant derivative with respect to index k. In the FullForm we see that CovariantD of a single tensor is an inert expression. In the display it is prefixed with a semi-colon.

```
In[7]:= Td[i]
        CovariantD[%, k]
        % // FullForm

Out[7]= T_i

Out[8]= T_{i ;k}

  Out[9]//FullForm=
       CovariantD[Tensor[T, List[Void], List[i]], k]
```

Multiple covariant derivative indices are preceded by a single semicolon in the default display. The differentiations are performed from the left to the right. This appears to be the convention in most texts. Tensorial was modified in Nov 2006 to follow this convention. The b index will be differentiated first and then the c index.

```
In[10]:= CovariantD[Td[a], {b, c}]
```

$$\text{Out[10]= } T_{a;b\,c}$$

An alternative display mode is to use the ∇ symbol.

```
In[11]:= SetCovariantDisplay["DelMode"]
         CovariantD[Td[a], {b, c}]
```

$$\text{Out[12]= } \nabla_{b\,c}\, T_a$$

If we write this as a nested differentiation and use HoldOp to prevent the otherwise automatic unnesting, then we see that the indices appear in reverse order in the display. But the b index is still done first and then the c index according to composition.

```
In[13]:= CovariantD[CovariantD[Td[a], b], c] // HoldOp[CovariantD]
```

$$\text{Out[13]= } \nabla_c \nabla_b T_a$$

Normal linear and Liebnizian differentiation rules are applied.

```
In[14]:= {Su[i] + Tu[i], Su[i] Tu[j]}
         CovariantD[#, k] & /@ %
```

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

$$\text{Out[15]= } \left\{\nabla_k S^i + \nabla_k T^i,\ \nabla_k T^j\, S^i + \nabla_k S^i\, T^j\right\}$$

Symbols and numeric quantities are not differentiated.

```
In[16]:= 2 π a Su[i] Tu[j]
         CovariantD[%, k]
```

$$\text{Out[16]= } 2\,a\,\pi\,S^i\,T^j$$

$$\text{Out[17]= } 2\,a\,\pi\left(\nabla_k T^j\, S^i + \nabla_k S^i\, T^j\right)$$

The flavor must be on the covariant index when differentiating flavored expressions.

```
In[18]:= 2 π a / (1 + ν) Su[i] Tu[j] // ToFlavor[red]
         CovariantD[%, red@k]
```

$$\text{Out[18]= } \frac{2\,a\,\pi\,S^i\,T^j}{1+\nu}$$

$$\text{Out[19]= } \frac{2\,a\,\pi\left(\nabla_k T^j\, S^i + \nabla_k S^i\, T^j\right)}{1+\nu}$$

A higher order derivative in the semicolon mode..

```
In[20]:= SetCovariantDisplay["SemicolonMode"]
         Su[i] Tu[j]
         CovariantD[%, {m, n}]
```

Out[21]= $S^i\ T^j$

Out[22]= $S^i{}_{;n}\ T^j{}_{;m} + S^i{}_{;m}\ T^j{}_{;n} + T^j{}_{;mn}\ S^i + S^i{}_{;mn}\ T^j$

```
In[23]:= CovariantD[Su[i], a] Tu[j]
         CovariantD[%, {m, n}]
```

Out[23]= $S^i{}_{;a}\ T^j$

Out[24]= $S^i{}_{;an}\ T^j{}_{;m} + S^i{}_{;am}\ T^j{}_{;n} + S^i{}_{;a}\ T^j{}_{;mn} + S^i{}_{;amn}\ T^j$

CovariantDerivatives of expressions with base indices or partial derivatives are not allowed. Instead, the covariant derivative of the abstract tensor should be taken, expanded with `ExpandCovariantD` and only then expanded to base indices.

```
In[25]:= Td[1]
         CovariantD[%, a]
         % // ExpandCovariantD[{x, δ, g, Γ}, {i}]
```

Out[25]= T_1

Out[26]= $T_{1;a}$

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

Out[27]= $Aborted

Covariant derivatives of scalar tensors are converted to partial derivatives when there is a single differentiation index.

```
In[28]:= Tensor[ϕ]
         CovariantD[%, i]
```

Out[28]= ϕ

Out[29]= $\phi_{,i}$

See `SetScalarSingleCovariantD` for handling sequential differentiations or disabling this feature.

Restore settings.

```
In[30]:= ClearTensorShortcuts[{{S, T}, 1}, {{S, T}, 2}]
```

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

See `ExpandCovariantD` for more examples.