Core Glossary DiracQ.m

In[3]:=
```
SetDirectory[NotebookDirectory[]];
Get["DiracQ_V0.m"];
```

An alphabetical list of commands in DiracQ followed by their meaning and usage.

```
? DiracQ`*
```

▼ DiracQ`

AddOperator	Operators
AllSymbols	Organize
anticommutator	OrganizedExpression
AntiCommutator	OrganizedProduct
AntiCommutatorDefinition	OrganizeQ
ApplyDefinition	p
b	PositionQ
b†	ProductQ
commutator	PushOperatorLeft
Commutator	PushOperatorRight
CommutatorDefinition	q
CommutatorRule	QCoefficient
CommuteParts	SecondaryOperators
Decomposition	SimplifyQ
DeleteOperator	StandardOrderQ
DiracQPalette	TakeCPart
DropQ	TakeQPart
Evaluation	TakeSummand
f	Vacuum
FullOrganize	x
function	X
f†	y
Humanize	z
Identical	δ
J	ϵ
n	σ
NCcross	†
OperatorProduct	$\hbar$

AddOperator

AddOperator allows users to expand the number of symbols that can be specified as noncommutative objects. The argument of AddOperator is the symbol that represents the new operator. Algebraic relations for new operators such as basic commutators, anticommutators, and products must be defined by the user. To define a basic commutator for two non commuting operators α and β, input CommutatorDefinition[α, β] := _ , where the blank is the definition of the commutator of α and β. AntiCommutatorDefinition[α, β] is the equivalent function for anticommutators and OperatorProduct[α, β] is the equivalent definition for operator products. Any number of such definitions can be input. Inputs using undefined variables are accepted. If a definition is called that has not been input by the user, the output will read ' Null'. The mutual consistency of different algebraic relations of a given symbol must be guaranteed by the user. DiracQ will use the stated properties as and when it finds an expression where it can be applied. Thus given inconsistent rules, the results of DiracQ will also be inconsistent.

- **Example**

```
AddOperator[θ]

Please enter all neccessary basic commutation
  and anticommutation relations. For help type ?AddOperator

CommutatorDefinition[θ[i_], θ[j_]] := ⅈ θ[i + j];

Commutator[θ[i], θ[j]]

ⅈ θ[i + j]
```

AllSymbols

All Symbols is an option setting which specifies that every non-numerical symbol will be viewed by every DiracQ function as a noncommutative object. By default DiracQ treats symbols as objects that commute unless otherwise specified. Activating All Symbols ensures that the manipulations performed by DiracQ proceed under the assumption that numbers commute but other symbols do not. All Symbols is selected as an option from the DiracQ Palette under the section "Active Operators".

- **Example**

In the DiracQ Palette activate All Symbols

```
Commutator[A ** B, C ** D]

A ** B ** C ** D - C ** D ** A ** B
```

anticommutator

anticommutator is the function used to specify an unknown or otherwise unevaluated anticommutator of only two elements. (Most users can ignore this command.)

- **Example**

In the DiracQ Palette set Apply Definition to false and activate Fermionic Annhilation and Creation Operators (f and f†):

```
AntiCommutator[f[a], f†[b]]

anticommutator[f[a], f†[b]]
```

The anticommutator remains unevaluated because Apply Definition has been set to false.

AntiCommutator

AntiCommutator is used to calculate the anticommutators of expressions. AntiCommutator[A, B] is defined as AB + BA. AntiCommutator accepts two arguments seperated by a comma. The arguments of AntiCommutator can be long and complex.

■ Example 1

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†):

```
AntiCommutator[f[a], f†[b]]
```

```
δ[a, b]
```

■ Example 2

In this example we take the anticommutator of a triple sum of destruction operators with a single creation operator. Notice that the fixed inded "k" is equated to the three running indices "i,j,l" one after the other.

$$\text{AntiCommutator}\left[2\sum_{i}\sum_{j}\sum_{l}\text{f[i] ** f[j] ** f[l], f†[k]}\right]$$

$$2\sum_{i}\sum_{j}\text{f[i] ** f[j]} - 2\sum_{i}\sum_{l}\text{f[i] ** f[l]} + 2\sum_{j}\sum_{l}\text{f[j] ** f[l]}$$

AntiCommutatorDefinition

AntiCommutatorDefinition is the function through which the anticommutators of symbols are defined. This function is used when adding operators through the AddOperator Function.

■ Example

```
AddOperator[θ]
```

```
Please enter all neccessary basic commutation
  and anticommutation relations. For help type ?AddOperator
```

```
AntiCommutatorDefinition[θ[i_], θ[j_]] := δ[i, j] θ[j]
```

```
AntiCommutator[θ[i], θ[j]]
```

```
δ[i, j] ** θ[j]
```

ApplyDefinition

ApplyDefinition is an option in the DiracQ Palette that specifies whether or not known properties of operators should be applied or not. When Apply Definition is set to True properties such as commutator, anticommutators, and products of operators will be evaluated. When Apply Definition is set to false commutators, anticommutators, and products will remain unevaluated.

■ Example

In the DiracQ Palette set Apply Definition to True and activate Bosonic Annhilation and Creation Operators (b and b†).

```
Commutator[b[i], b†[j]]
```

```
δ[i, j]
```

Now set Apply Definition to False.

```
Commutator[b[i], b†[j]]
```

```
commutator[b[i], b†[j]]
```

b

b is the Bosonic annihilation operator. This operator requires one index denoting site, and a second optional index can be used to denote spin or species. Also included is the Bosonic number operator, represented by n_b. The argument scheme for the number operator is identical to that of the annihilation operator. For more information on site index see below.

Example

In the DiracQ Palette activate Bosonic Annhilation and Creation Operators (b and b†).
The b operator obeys the canonical commutation relation for Bosons.

```
Commutator[b[1], b†[1]]
```

```
1
```

Bra

Bra[x] represents the bra vector <x| using Dirac’s notation

Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†) and Bra and Ket Vectors.

```
Bra[A] = Bra[Vacuum] ** f[1]
```

```
Bra[Vacuum] ** f[1]
```

In the calculation below we use the important function ProductQ that is described more fully below. This function uses all the non commutative properties supplied by the user and implements them to give an easily readable and correct final result. In this calculation, the “f” fermion at site 1 that is present in Bra[A] is destroyed by f†[1] acting to its left.

```
ProductQ[Bra[A], f†[1]]
```

```
Bra[Vacuum]
```

b†

b† is the Bosonic creation operator. This operator requires one index denoting site, and a second optional index can be used to denote spin or species. Also included is the bosonic number operator, represented by n_b. The argument scheme for the number operator is identical to that of the annihilation operator. For more information on site index see Site Index below. The dagger symbol is created by entering ESCdgESC or clicking the appropriate button on the DiracQ palette.

Example

In the DiracQ Palette activate Bosonic Annhilation and Creation Operators (b and b†).
The b operator obeys the canonical commutation relation for bosons.

```
Commutator[b[1], b†[1]]
```

```
1
```

commutator

commutator is the function used to specify an unknown or otherwise unevaluated commutator of only two elements. (Most users can ignore this command.)

Example

In the DiracQ Palette set Apply Definition to false and activate Bosonic Annhilation and Creation Operators (f and f†):

```
Commutator[b[i], b†[j]]
```

```
commutator[b[i], b†[j]]
```

The commutator remains unevaluated because Apply Definition has been set to false. If we do not set Aply Definition is set to true, the default setting, the result we obtain would use the known commutator of Bosonic operators.

In the DiracQ Palette set Apply Definition to true:

```
Commutator[b[i], b†[j]]
```

```
δ[i, j]
```

Commutator

Commutator is used to calculate the commutators of expressions. Commutator[A, B] is defined as AB - BA. Commutator accepts two arguments seperated by a comma. The arguments of Commutator can be long and complex.

Examples

Example 1

In the DiracQ Palette activate Bosonic Annhilation and Creation Operators (b and b†):

```
Commutator[b[a], b†[b]]
```

```
δ[a, b]
```

In the example below, the c number function u[i,j] does not take part in the commutator, but its argument gets its value from the Kronecker Delta function of the external index “k” with one of the two internal indices “i,j”.

$$\text{Commutator}\left[2\sum_i\sum_j u[i, j]\; b[i] ** b[j],\; b^\dagger[k]\right]$$

$$2\sum_i u[i, k] ** b[i] + 2\sum_j u[k, j] ** b[j]$$

CommutatorDefinition

CommutatorDefinition is the function through which the commutators of symbols are defined. This function is used when adding operators through the AddOperator Function.

▪ Example

```
AddOperator[θ]

Please enter all neccessary basic commutation
  and anticommutation relations. For help type ?AddOperator

CommutatorDefinition[θ[i_], θ[j_]] := δ[i, j] θ[j]

Commutator[θ[i], θ[j]]

δ[i, j] ** θ[j]
```

CommuteParts

CommuteParts[expr,{list1},{list2}] will reverse the order of the noncommuting objects specified by the lists list1 and list2. list1 and list2 specify lists of operators found in expr. Operators are specified by numerical ordering of the operators found in expr. To permute the second thru fourth operators found in expr with the fifth operator found in expr, list1 would be {2,4}, and list2 would be {5}.

▪ Example

In the DiracQ Palette activate Canonical Position and Momentum Operators.
The following input will permute the second thru fourth operators found in the expression with the fifth operator found in the expression. Any nonzero commutators will be accounted for.

```
CommuteParts[p[i] ** p[j] ** q[i] ** q[j] ** q[k], {2, 4}, {5}]

-ⅈ (ℏ δ[k, j]) ** p[i] ** q[i] ** q[j] + p[i] ** q[k] ** p[j] ** q[i] ** q[j]
```

Only operators are counted for ordering purposes. (The assumption here is that the "All Symbols" option is not used in the pallette- but only relevant symbols are declared to be operators). Therefore, in the example below, p[i] is specified by {1} and q[j] by {2}.

```
CommuteParts[5 a ∑_i p[i] ** q[j], {1}, {2}]

5 ∑_i a ** q[j] ** p[i] - 5 ⅈ ∑_i a ℏ δ[i, j]
```

Decomposition

The panel Decomposition on the DiracQ Palette gives the user two choices, Commutator or Anticommutator. The default setting is to decompose everything into commutators. Making a choice of decomposition allows the user to control whether the commutators of composite symbols will be decomposed into basic commutators or anticommutators, using appropriate standard rules. Not all combinations of operators can be decomposed entirely into basic commutators or anticommutators. Changing the setting of decomposition has no effect so long as ApplyDefinition is set to true. When ApplyDefinition is set to false, the decomposition settings make a difference in the results.

▪ Example

In the DiracQ Palette set Apply Definition to False and activate All Symbols:

```
Commutator[a ** b, c]
```

```
a ** commutator[b, c] - commutator[c, a] ** b
```

Now in the DiracQ Palette set Decomposition to AntiCommutator:

```
Commutator[a ** b, c]
```

```
a ** anticommutator[b, c] - anticommutator[c, a] ** b
```

DeleteOperator

DeleteOperator will remove a user defined operator from the list of possible operators. The argument of delete operators is the symbol by which the operator is represented.

■ Example

```
AddOperator[θ]
```

```
Please enter all neccessary basic commutation
  and anticommutation relations. For help type ?AddOperator
```

Calling Operators will display all currently active operators.

```
Operators
```

```
{θ}
```

```
DeleteOperator[θ]
```

```
{}
```

```
Operators
```

```
{}
```

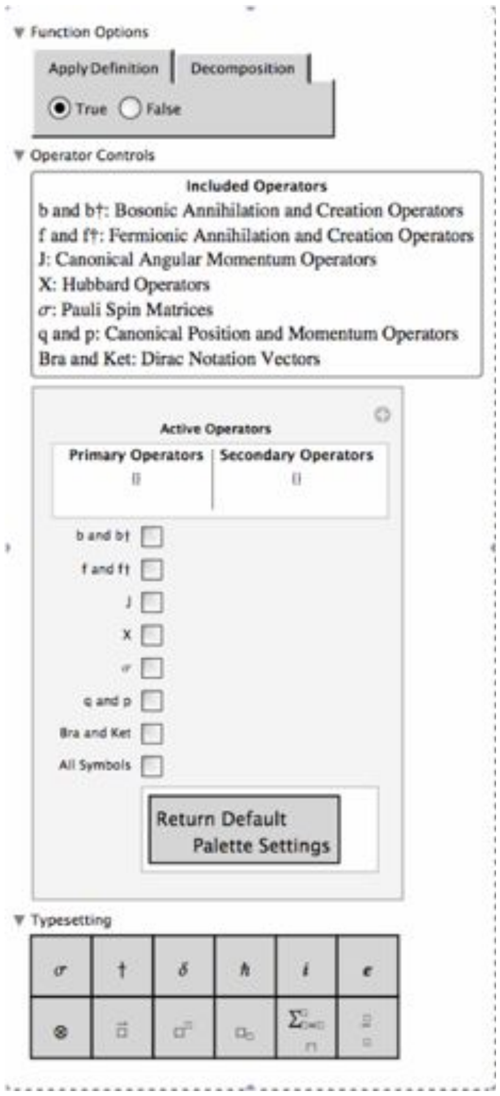
DiracQPalette

DiracQPalette will open the DiracQ Palette.

Essential to the operation of the package is the DiracQPalette. This palette allows users to control the options of package functions, designate symbols as operators, and input some of the special symbols and functions used frequently when operatoring the package. To open the palette, input the function DiracQPalette. The palette should also open automatically when the package is first loaded.

```
In[5]:= SetOptions[DiracQPalette, Magnification → 1.25]
```

A new notebook should open that is identical to the notebook shown below.



Function Options

Apply Definition | Decomposition

[✓ True] False

Operator Controls

Included Operators

b and b†: Bosonic Annihilation and Creation Operators
f and f†: Fermionic Annihilation and Creation Operators
J: Canonical Angular Momentum Operators
X: Hubbard Operators
σ: Pauli Spin Matrices
q and p: Canonical Position and Momentum Operators
Bra and Ket: Dirac Notation Vectors

Active Operators

Primary Operators	Secondary Operators
{}	{}

b and b† ☐
f and f† ☐
J ☐
X ☐
σ ☐
q and p ☐
Bra and Ket ☐
All Symbols ☐

Return Default Palette Settings

Typesetting

The DiracQ package contains a collection of commonly used quantum mechanical operators for which algebraic relations such as commutators, anticommutators, and products are known. The functions of the package recognize certain symbols as operators and use these algebraic relations to manipulate and evaluate user input. To avoid confusion the package requires explicit instructions from the user to view any certain symbol as an operator. By default no symbols are viewed as operators. This prevents users from making assumptions about the status of a symbol.

The DiracQ palette contains a section titled "Operator Controls". The purpose of this section is to control how the functions of the package view a symbol. This section contains a list of the operators that are included for use in the package and the symbols used to represent them. For a symbol on this list to be viewed as an operator it must first be activated by the user. This is done by locating the desired symbol(s) in the subsection titled "Active Operators" and checking the box to the right of the symbol(s). The symbol should now populate the list of primary operators. Any additional secondary operators which are composites of the primary operators will be found in the list "Secondary Operators". Once a symbol is activated that symbol will be viewed as an operator, or “q” number, by all of the DiracQ functions. The symbol will also be given the special algebraic properties of the quantum mechanical operator to which it corresponds. The same symbol when inactivated will be viewed as a “c” number with no special algebraic properties.

DropQ

DropQ[expr,n] gives expr with the operators specified by n dropped. The specification of operators is identical to that used by the Drop function to specify which elements of a list to drop. The operators are specified by the order in which they appear in expr.

■ Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (p and q).
The notation used by DropQ is very similar to that used by the Mathematica function Drop.

```
? Drop
```

Drop[*list*, *n*] gives *list* with its first *n* elements dropped.
Drop[*list*, −*n*] gives *list* with its last *n* elements dropped.
Drop[*list*, {*n*}] gives *list* with its n^{th} element dropped.
Drop[*list*, {*m*, *n*}] gives *list* with elements *m* through *n* dropped.
Drop[*list*, {*m*, *n*, *s*}] gives *list* with elements *m* through *n* in steps of *s* dropped.
Drop[*list*, seq_1, seq_2, …] gives a nested
list in which elements specified by seq_i have been dropped at level *i* in *list*. ≫

Therefore, the input below will drop the first operator in the expression.

```
DropQ[-ⅈ (ⅇ^2 Z ℏ) ** q[i, y] ** p[i, z] ** q[i, y], {1}]
```

```
-ⅈ (ⅇ^2 Z ℏ) ** p[i, z] ** q[i, y]
```

The input below will drop the first and second operators.

```
DropQ[-ⅈ (ⅇ^2 Z ℏ) ** q[i, y] ** p[i, z] ** q[i, y], 2]
```

```
-ⅈ (ⅇ^2 Z ℏ) ** q[i, y]
```

The assumption here is that the “All Symbols” option is not used in the pallette- but only relevant symbols are declared to be operators. In case “All Symbols” are activated in the Palette, DropQ will regard the prefactors (such as $(ⅇ^2 Z \hbar)$) in this case to be operators too and give a correspondingly different output.

Evaluation

Evaluation is an option for the Kronecker-δ function. If Evaluation is set to Identical the δ will evaluate to zero unless both arguments are indentical.

■ Example

The kronecker delta function by default does not evaluate two different symbols to be zero unless it knows the symbols to be numerically different.

```
δ[i, j]
```

```
δ[i, j]
```

```
i = 1;
j = 2;
δ[i, j]
Clear[i, j];
```

```
0
```

When evaluation is set to identical the kronecker delta will evaluate to zero unless the symbols are identical.

```
δ[i, j, Evaluation -> Identical]
```

```
0
```

```
δ[i, i, Evaluation -> Identical]
```

```
1
```

f

f is the Fermionic annihilation operator. This operator requires one index denoting site, and a second optional index can be used to denote spin or species. Also included is the Fermionic number operator, represented by n_f. The argument scheme for the number operator is identical to that of the annihilation operator. For more information on site index see Site Index below.

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†).
The f operator obeys the canonical anticommutation relation for fermions.

```
AntiCommutator[f[1], f†[1]]
```

```
1
```

FullOrganize

FullOrganize is an extension of the Organize function. FullOrganize incorporates Organize, OrganizeQ, and OrganizedProduct into one function. Therefore FulllOrganize takes an expression, organizes it into the DiracQ organized notation, orders the operators, and applies product rules to the list of operators. See Also: Organize, OrganizeQ, OrganizedProduct

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†) and Bosonic Annhilation and Creation Operators (b and b†).

The two examples below demonstrate the structure of expressions in DiracQ. The first example has no summations in it, three c-number prefactors (5 a c) and a pair of Fermi and a pair of Bose operators. Reading from right to left, these are arranged as shown with the q-numbers in the last bracket, the c numbers in the next bracket, the summed indices (null in this case) in the next bracket and the numerical coefficient in the first bracket. The second example has three summed indices and a more complicated c-number part that are pulled out in the given format. Within the q-number bracket, there is a canonical ordering with Bosons to the left of Fermions etc. (If we do not activate the Fermion or Boson operators in the palette, the FullOrganize function will treat f and b as c-numbers as well.)

```
FullOrganize[5 a c f[i] ** b[i] ** f[j] ** b[k]]
```

```
{{5, {}, {a c}, {b[i], b[k], f[i], f[j]}}}
```

FullOrganize$\left[\sum_{i}\sum_{k}\sum_{j}\right.$ 5 a c u[i, j, k] f[i] ** b[i] ** f[j] ** b[k]$\left.\right]$

```
{{5, {i, j, k}, {a c u[i, j, k]}, {b[i], b[k], f[i], f[j]}}}
```

function

function is a label used by the Organize function to denote a function of operators that can not be decomposed simply into polynomials. The first argument is the function, and the second argument is the operators on which the function depends. This function is not intended to be manipulated by the user but is used in certain internal computations. The first argument of function is the function containing an operator, and the second argument are the operators of which that functiond depends. In general DiracQ does not know the algebraic properties of non-polynomial function of operators. The user can input algebraic properties of functions if necessary using the same methods as inputing algebraic properties of user created operators.

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†) and Bosonic Annhilation and Creation Operators (b and b†).

The example below shows that the power function of f†[i] below is recognized to contain an operator, and is therefore organized as an operator using the function notation.

Organize$\left[e^{f\dagger[i]}\right]$

$\left\{\left\{1, \{\}, \{1\}, \left\{\text{function}\left[e^{f\dagger[i]}, \{f\dagger[i]\}\right]\right\}\right\}\right\}$

DiracQ does not know what the commutator of this function of f†[i] and other fermionic operators is.

Commutator$\left[e^{f\dagger[i]}, f[i]\right]$

```
Error: Unknown commutator called!
```

We are able to teach DiracQ how to perform this commutator by using the CommutatorDefinition function, as shown below.

```
CommutatorDefinition[function[a_, {f†[i_]}], f[j_]] := a ** f[j] - f[j] ** a
```

Commutator$\left[e^{f\dagger[i]}, f[i]\right]$

$e^{f\dagger[i]} ** f[i] - f[i] ** e^{f\dagger[i]}$

f†

f† is the Fermionic creation operator. This operator requires one index denoting site, and a second optional index can be used to denote spin. Also included is the Fermionic number operator, represented by n_f. The argument scheme for the number operator is identical to that of the annihilation operator. For more information on site index see Site Index in the DiracQ Glossary. The dagger symbol is created by entering ESCdgESC or clicking the appropriate button on the DiracQ palette.

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†).
The f† operator obeys the canonical anticommutation relation for fermions

```
AntiCommutator[f[1], f†[1]]
```

```
1
```

Humanize

Humanize is the functional opposite of Organize. Humanize takes a nested list of terms organized according to the method of the package and yields output of familiar mathematical forms. Humanize only reconizes input that is in the format of the output of the Organize function. For more information on the organization system of the package see Organize

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†).

The Organize function decomposes an expression containing a variety of terms and algebraic forms and converts it into a nested list. The Organize function is used by a majority of the DiracQ functions to convert input into a form that can be easily manipulated.

```
Organize[2 a ∑_i f[i] + 3 b ∑_j f†[j]]
```

```
{{2, {i}, {a}, {f[i]}}, {3, {j}, {b}, {f†[j]}}}
```

Humanize is used to return the nested list structure used to manipulate expressions to standard mathematical forms.

```
Humanize[%]
```

```
2 ∑_i a ** f[i] + 3 ∑_j b ** f†[j]
```

Identical

Identical is an option setting for the Kronecker δ option Evaluation.If Evaluation is set to Identical the δ will evaluate to zero unless both arguments are indentical. From more information on the Kronecker δ see δ

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†).
The Kronecker δ will only evaluate to zero if two symbols are known to be numerically different.

```
δ[1, 0]
```

```
0
```

```
δ[i, j]
```

```
δ[i, j]
```

```
Clear[i, j];
i = 1;
j = 0;
δ[i, j]
```

```
0
```

The option Evaluation allows users to specify that unless two symbols are identical the Kronecker δ should evaluate to zero.

```
δ[i, j, Evaluation → Identical]
```

```
0
```

J

J is the canonical angular momentum operator. This operator requires at least two arguments. The first is site index and the second is coordinate direction. An optional third argument is used to denote different species. Also included are the angular raising and lowering operators, denoted by J^{Plus} and J^{Minus} respectively. The raising and lowering operators accept only one argument corresponding to site, and if provided, a second argument taken to represent species. For more information on site index see Site Index. The action of the angular momentum operators on standard basis states will be implemented later, for now only their commutation rules are available.

Example

In the DiracQ Palette activate the Canonical angular momentum operator (J).
The J operator obeys the canonical commutation relations for angular momentum operators

```
Commutator[J[i, x], J[j, y]]
```

```
i (ℏ δ[i, j]) ** J[i, z]
```

```
Commutator[J[i, z], J[j, y]]
```

```
-i (ℏ δ[i, j]) ** J[i, x]
```

The third argument will be taken to represent species. In the following example we take the commutator of two angular momentum operators that act on different species, represented by g and h.

```
Commutator[J[i, x, g], J[j, y, h]]
```

```
i (ℏ δ[g, h] δ[i, j]) ** J[i, z]
```

Using the angular momentum raising and lowering operators the first argument is taken to be site index, and the second argument is used to denote different species.

Commutator$\left[J^{\text{Plus}}[i, a], J^{\text{Minus}}[j, b]\right]$

```
2 (ℏ δ[a, b] δ[i, j]) ** J[i, z, a]
```

Ket

Ket[x] represents a ket vector |x> using Dirac notation

Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†) and Bra and Ket Vectors.

```
Ket[A] = f†[1] ** Ket[Vacuum]
```

```
f†[1] ** Ket[Vacuum]
```

In the calculation below we use the important function ProductQ that is described more fully below. This function uses all the non commutative properties supplied by the user and implements them to give an easily readable and correct final result. In this calculation, the “f” fermion at site 1 that is present in Ket[A] is destroyed by f[1] acting to its left.

```
ProductQ[f[1], Ket[A]]
```

```
Ket[Vacuum]
```

n

n is the number operator. To specify the number operator for bosons use n_b and for fermions use n_f.

Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†) and Bosonic Annhilation and Creation Operators (b and b†)

```
n_f[a]
```

```
f†[a] ** f[a]
```

```
n_b[a]
```

```
b†[a] ** b[a]
```

```
Commutator[n_b[a], b†[a]]
```

```
b†[a]
```

NCcross

Non Commutative cross product of two 3dimensional vectors retaining the order of the operators.

Example

As an example consider the orbital angular momentum operator $\vec{L}$ and a further cross product with the position vector $\vec{q}$. Note that $\vec{q}$ and $\vec{p}$ need to be specified with an argument, arbitrarily chosen here to be i.

```
L⃗ = NCcross[q⃗[i], p⃗[i]]
```

```
{q[i, y] ** p[i, z] - q[i, z] ** p[i, y],
 -q[i, x] ** p[i, z] + q[i, z] ** p[i, x], q[i, x] ** p[i, y] - q[i, y] ** p[i, x]}
```

```
v1 = NCcross[q⃗[i], L⃗]
```

```
{q[i, y] ** (-q[i, y] ** p[i, x]) - q[i, z] ** (-q[i, x] ** p[i, z]) +
  q[i, y] ** q[i, x] ** p[i, y] - q[i, z] ** q[i, z] ** p[i, x],
 -q[i, x] ** (-q[i, y] ** p[i, x]) + q[i, z] ** (-q[i, z] ** p[i, y]) -
  q[i, x] ** q[i, x] ** p[i, y] + q[i, z] ** q[i, y] ** p[i, z],
 q[i, x] ** (-q[i, x] ** p[i, z]) - q[i, y] ** (-q[i, z] ** p[i, y]) +
  q[i, x] ** q[i, z] ** p[i, x] - q[i, y] ** q[i, y] ** p[i, z]}
```

Here we see that the non commutative nature is redundant in defining $\vec{L}$ as such, since the canonical pairs $q[x]$ and $p[x]$ are kept away from each other already. However, in the next cross product, it is relevant since the canonical pairs are no longer separated. This function is used in the Runge Lenz vector definition as an example.
Further note: if one wants a neater answer with negative signs pulled out, we can map the DiracQ function SimplifyQ to each component of the output.

```
SimplifyQ /@ v1
```

```
{q[i, x] ** q[i, y] ** p[i, y] + q[i, x] ** q[i, z] ** p[i, z] -
  q[i, y] ** q[i, y] ** p[i, x] - q[i, z] ** q[i, z] ** p[i, x],
 -q[i, x] ** q[i, x] ** p[i, y] + q[i, x] ** q[i, y] ** p[i, x] + q[i, y] ** q[i, z] ** p[i, z] -
  q[i, z] ** q[i, z] ** p[i, y], -q[i, x] ** q[i, x] ** p[i, z] +
  q[i, x] ** q[i, z] ** p[i, x] - q[i, y] ** q[i, y] ** p[i, z] + q[i, y] ** q[i, z] ** p[i, y]}
```

OperatorProduct

OperatorProduct is the function through which the products of symbols are defined. OperatorProduct is used only when creating additional operators. For examples of the usage of OperatorProduct see

■ Example

```
AddOperator[θ]
```

```
Please enter all neccessary basic commutation
  and anticommutation relations. For help type ?AddOperator
```

```
Operators
```

```
{θ}
```

```
OperatorProduct[θ[i_], θ[j_]] := (1 - δ[i, j]) θ[i] ;
```

```
ProductQ[θ[i], θ[j]]
```

```
-δ[i, j] ** θ[i] + θ[i]
```

Any type of OperatorProduct rule or operator symbol can be used.

```
AddOperator[u]
```

```
Please enter all neccessary basic commutation
  and anticommutation relations. For help type ?AddOperator
```

```
OperatorProduct[u[i_], u[j_]] := I u[i + j]
```

```
ProductQ[u[1], u[2]]
```

```
ⅈ u[3]
```

Operators (command)

Operators is the list of symbols that are currently being recognized as operators.

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†) and Bosonic Annhilation and Creation Operators (b and b†).

```
Operators
```

```
{b, b†, f, f†}
```

Organize

Organize is the function that enables the DiracQ package to understand and manipulate user input. Organize takes a mathematical expression as input and yields a nested list that contains the atoms of the input ordered according to their properties. Numbers, summed indices, c numbers, and q numbers are separated into groups. Each term of the input separated by plus sign constitutes a separate list of items in the output. For more information see the Explanation of Form Appendix in the DiracQ Writeup notebook.

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†).

Below is a diagram showing in general how input is decomposed by the organize function. # indicates any number as recognized by mathematica, c# is a nonnumerical symbol that is not recognized as an operator, q# is any noncommutative symbol, or operator, as recognized by the DiracQ package, and index indicates a nonnumerical index over which an expression is being summed.

$$\text{Organize}\left[(\#)\sum_{\text{index}}(c\,\#)*(q\,\#)\right]$$

$$= \{\{\sharp, \{\text{index}\}, \{c\,\sharp\}, \{q\,\sharp\}\}\}$$

$$\text{Organize}\left[(\sharp_1)\sum_{\text{index}_1}(c\,\sharp_1) * (q\,\sharp_1) + (\sharp_2)\sum_{\text{index}_2}(c\,\sharp_2) * (q\,\sharp_2)\right]$$
$$= \{\{\sharp_1, \{\text{index}_1\}, \{c\,\sharp_1\}, \{q\,\sharp_1\}\}, \{\sharp_2, \{\text{index}_2\}, \{c\,\sharp_2\}, \{q\,\sharp_2\}\}\}$$

Below is an example of the organize function containing Fermionic annhilation and creation operators

$$\textbf{Organize}\left[5\sum_{i} a[i]\, f[i] + 2\sum_{j} b[j]\, f\dagger[j]\right]$$

```
{{5, {i}, {a[i]}, {f[i]}}, {2, {j}, {b[j]}, {f†[j]}}}
```

OrganizedExpression

OrganizedExpression is an option of several DiracQ functions which allows a user to input a preorganized expression into a function which normally accepts standard form input. This is normally done in the interest of time saving. To use preorganized input include the option setting OrganizedExpression -> True. This function is mainly used within the package when one function operates within another function, but it may be of use to users who are dealing with long calculations. See Standard OrderQ for information on the use of the StandardOrderQ function.

■ Example

In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†) and Bosonic Annhilation and Creation Operators (b and b†).

SimplifyQ normally excepts standard input

```
A = f[i] ** b[i] ** f[j];
B = Organize[A];
```

```
StandardOrderQ[A]
```

```
b[i] ** f[i] ** f[j]
```

When organized input is used with StandardOrderQ it remains unchanged

```
StandardOrderQ[B]
```

```
{{1, {}, {1}, {f[i], b[i], f[j]}}}
```

When the OrganizedExpression option is set to True we see that the correct answer is obtained, but that in remains in organized form.

```
StandardOrderQ[B, OrganizedExpression → True]
```

```
{{1, {}, {1}, {b[i], f[i], f[j]}}}
```

If we use the Humanize function to return it to standard form we see readily that the manipulation has been performed.

```
Humanize[%]
```

```
b[i] ** f[i] ** f[j]
```

OrganizedProduct

OrganizedProduct is a function used within the package that is not relevant to most users. OranizedProduct takes organized input and simplifies the operators by evaluating products if possible. Output is also organized.

■ Example

OrganizedProduct is used inside the package in certain simplifications, but is of little use to most users who will find Organize to achieve the goal of rewriting input operators in a systematically neater form.

```
OrganizedProduct[Organize[-ⅈ (ⅇ² Z ℏ) ** q[i, y] ** q[i, y] ** p[i, z]]]
```

```
{{-ⅈ, {}, {ⅇ² Z ℏ}, {q[i, y], q[i, y], p[i, z]}}}
```

If we drop the initial Organize above, we get errors. With Organize as shown, it simply regurgitates the input and so is redundant- as mentioned above.

OrganizeQ

OrganizeQ is a function used within the package that is not relevant to most users.
OrganizeQ takes organized input and rearranges the operators according to a standardized order. OrganizeQ is a subfunction of the FullOrganize function. Output is also organized.

■ Example

OrganizeQ is used inside the package in certain simplifications, but is of little use to most users who will find Organize to achieve the goal of rewriting input operators in a systematically neater form.

```
OrganizeQ[Organize[-ⅈ (ⅇ² Z ℏ) ** q[i, y] ** q[i, y] ** p[i, z]]]
```

```
{{-ⅈ, {}, {ⅇ² Z ℏ}, {q[i, y], q[i, y], p[i, z]}}}
```

If we drop the initial Organize above, we get errors. With Organize as shown, it simply regurgitates the input and so is redundant- as mentioned above.

p

p is the canonical momentum operator. This operator can be called with one argument, assumed to be the particle number or the site index. If two arguments are given, the second argument will be taken to be coordinate direction. Also included is the 3 dimensional canonical momentum vector, represented by OverVector[p], or $\vec{p}$

■ Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (q and p).

```
Commutator[q[i], p[i]]
```

```
ⅈ ℏ
```

```
Commutator[q[i, x], p[i, x]]
```

```
ⅈ ℏ
```

```
Commutator[q[i, x], p[i, y]]
```

```
0
```

```
p⃗[i]
{p[i, x], p[i, y], p[i, z]}
OverVector[p][i]
{p[i, x], p[i, y], p[i, z]}
```

PositionQ

PositionQ[expr, pattern] gives a list of the positions of an operator matching pattern appear in expr. The position given is the position of the operator relative to other operators in expr only.

■ Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (p and q).

The output below shows that in the first term of the expression p[i, z] is the second operator and in the second term of the expression p[i,z] is the third operator.

```
PositionQ[
 -ⅈ (ⅇ² Z ℏ) ** q[i, y] ** p[i, z] ** q[i, y] - ⅈ (ⅇ² Z ℏ) ** q[i, z] ** q[i, z] ** p[i, z], p[i, z]]
{{1, 2}, {2, 3}}
```

ProductQ

ProductQ gives the product of two expressions involving terms that are noncommutative objects. ProductQ should be used in place of the standard Mathematica function NonCommutativeMultiply for combining expressions. ProductQ can be called as a function with two arguments or as the CircleTimes symbol ESCc*ESC used between two expressions. Operator product definitions will be applied by default. Settings are specified through the SetSession function.

■ Example

ProductQ is far more useful for most quantum applications than NonCommutativeMultiply. However, this does not mean that should always be used in place of NonCommutativeMultiply. NCM is useful to seperate terms that must not be reorganized by Mathematica' s sorting. ProductQ is useful for combining larger expressions and appyling the product definitions of operators. The default output is ordered according to standard ordering.

```
A = ∑_i t[i] σ[i, y] ** σ[i, x];
B = σ[k, y];
A ⊗ B
-ⅈ ∑_i t[i] ** σ[i, z] ** σ[k, y]
```

This function has built in useful properties such as the Pauli principle, so that the square of a destruction or creation operator for Fermions vanishes. In the DiracQ Palette activate Fermionic Annhilation and Creation Operators (f and f†)

```
ProductQ[f[i], f[i]]
0
```

```
f[i] ⊗ f[i]
```

```
0
```

```
ProductQ[f[j], f[i]]
```

```
-f[i] ** f[j]
```

```
ProductQ[u[i] f[i] ** f†[i], v[j] f[k]]
```

```
(u[i] v[j]) ** f[k] - (u[i] v[j]) ** n_f[i] ** f[k]
```

In the first example a simple NonCommutativeMultiply (i.e. **) gives the same result as the ProductQ. In the last example, the power of ProductQ is seen, it pulls apart the c-number parts and then applies all given relations to the q-number part, giving an easy to read result.

PushOperatorLeft

PushOperatorRight[expr,pattern] will move the operator matching pattern to the left of all other operators in every term in expr. Commutators are accounted for.

■ Example

Sometimes the order of operators output by the package is not appropriate for a desired manipulation. PushOperatorLeft provides a powerful and convenient way to move a given operator to the left of all other operators in an expression, while picking up the commutator terms obtained in the reordering.

In the DiracQ Palette activate Canonical Position and Momentum Operators (p and q).

```
PushOperatorLeft[
 -ⅈ (ⅇ^2 Z ℏ) ** q[i, y] ** q[i, y] ** p[i, z] - ⅈ (ⅇ^2 Z ℏ) ** q[i, z] ** q[i, z] ** p[i, z], p[i, z]]
```

```
2 (ⅇ^2 Z ℏ^2) ** q[i, z] - ⅈ (ⅇ^2 Z ℏ) ** p[i, z] ** q[i, y] ** q[i, y] -
 ⅈ (ⅇ^2 Z ℏ) ** p[i, z] ** q[i, z] ** q[i, z]
```

Here the operator p[i,z] is located in the expression and pushed all the way to the left.

PushOperatorRight

PushOperatorRight[expr,pattern] will move the operator matching pattern to the right of all other operators in every term in expr. Commutators are accounted for.

■ Example

This is provided as an alternative to PushOperatorLeft. Sometimes the order of operators output by the package is not correct for a desired manipulation. PushOperatorRight provides a powerful and convenient way to move a given operator to the right of all other operators in an expression.

In the DiracQ Palette activate Canonical Position and Momentum Operators (p and q).

```
PushOperatorRight[
 -ⅈ (ⅇ^2 Z ℏ) ** p[i, z] ** q[i, y] ** q[i, y] - ⅈ (ⅇ^2 Z ℏ) ** p[i, z] ** q[i, z] ** q[i, z], p[i, z]]
```

```
-2 (ⅇ^2 Z ℏ^2) ** q[i, z] - ⅈ (ⅇ^2 Z ℏ) ** q[i, y] ** q[i, y] ** p[i, z] -
 ⅈ (ⅇ^2 Z ℏ) ** q[i, z] ** q[i, z] ** p[i, z]
```

Here the operator p[i,z] is pushed all the way to the right.

q

q is the canonical position operator. This operator can be called with one argument, assumed to be the particle number or the site index. If two arguments are given, the second argument will be taken to be coordinate direction. Also included is the 3 dimensional canonical position vector, represented by OverVector[p], or $\vec{q}$.

■ Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (q and p).

```
Commutator[q[i], p[i]]
```

```
ⅈ ℏ
```

```
Commutator[q[i, x], p[i, x]]
```

```
ⅈ ℏ
```

```
Commutator[q[i, x], p[i, y]]
```

```
0
```

```
q⃗[i]
```

```
{q[i, x], q[i, y], q[i, z]}
```

```
OverVector[q][i]
```

```
{q[i, x], q[i, y], q[i, z]}
```

QCoefficient

QCoeffficient[expression, form] is the DiracQ equivalent of the function Coefficient in *Mathematica*. It scans the "expression" for terms containing a string of operators that match "form". It returns the c-number coefficient, with "form" being a q-number pattern. If several terms are found the output will be a sum of terms. Only exact matches of form are found. (A series of which "form" is a subpart will not be included).

■ Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (q and p) and Pauli Spin Matrices (σ).

QCoefficient will yield the coefficient of the operators requested.

```
QCoefficient[4 a[i] p[i] ** q[i], p[i] ** q[i]]
```

```
4 a[i]
```

QCoefficient can be used to scan more complicated input involving many terms as well.

$$\text{Term1} = \frac{a+b}{2} + \frac{a-b}{2}\,\sigma[1, z] ** \sigma[3, z] + \frac{c+d}{2}\,\sigma[1, x] ** \sigma[3, x] + \frac{c-d}{2}\,\sigma[1, y] ** \sigma[3, y];$$

$$\text{Term2} = \frac{a'+b'}{2} + \frac{a'-b'}{2}\,\sigma[2, z] ** \sigma[3, z] + \frac{c'+d'}{2}\,\sigma[2, x] ** \sigma[3, x] + \frac{c'-d'}{2}\,\sigma[2, y] ** \sigma[3, y];$$

```
product = Term1 ⊗ Term2;
Length[product]
```

```
64
```

We can scan product and find the coefficients of all of the terms that contain a single string of operators. For short output this can be done manually but becomes difficult with longer output. We see here that the expression product is quite long and contains for terms with the string of operators listed. The result of QCoefficient of q is therefore a sum of all these coefficients.

QCoefficient[product, σ[1, x] ** σ[2, y] ** σ[3, z]]

$$\frac{1}{4} i\, c\, c' + \frac{1}{4} i\, d\, c' - \frac{1}{4} i\, c\, d' - \frac{1}{4} i\, d\, d'$$

SecondaryOperators

SecondaryOperators of all the active composite operators, that is, all of the active operators that are combinations of more basic operators. The default list is $\{n_f, n_b, \sigma^{\text{Plus}}, \sigma^{\text{Minus}}, J^{\text{Plus}}, J^{\text{Minus}}, \vec{p}, \vec{q}\}$

▪ Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (q and p).

SecondaryOperators

$\{\vec{q}, \vec{p}\}$

SimplifyQ

SimplifyQ is analogous to the existing Mathematica Simplify function for expressions that contain noncommutative objects. All output of DiracQ function such as Commutator or ProductQ is already simplified in the manner performed by this function. This function is primarily used to simplify expressions that have either never been input into a DiracQ function or have been manipulated by the user and need to be simplified.

▪ Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (q and p).

The following expression “term1” from a harmonic oscillator problem was generated:

$$\mathbf{term1} = \frac{\sqrt{\frac{m\,\omega}{\hbar}} ** q[i]}{\sqrt{2}} - \frac{i\,\frac{1}{\sqrt{\frac{m\,\omega}{\hbar}}\,\hbar} ** p[i]}{\sqrt{2}};$$

Performing further manipulations on this expression by hand enables us to pull out a common factor and subtracting the original expression we form a new expression “term2” that should evaluate to zero.

$$\texttt{term2 = term1} - \frac{\sqrt{\frac{m\,\omega}{\hbar}}\left(-\frac{i\,p[i]}{m\,\omega}+q[i]\right)}{\sqrt{2}}$$

$$\frac{\sqrt{\frac{m\,\omega}{\hbar}} ** q[i]}{\sqrt{2}} - \frac{i\,\frac{1}{\sqrt{\frac{m\,\omega}{\hbar}}\,\hbar} ** p[i]}{\sqrt{2}} - \frac{\sqrt{\frac{m\,\omega}{\hbar}}\left(-\frac{i\,p[i]}{m\,\omega}+q[i]\right)}{\sqrt{2}}$$

This expression can be simplified using SimplifyQ since it does a full book keeping of all terms within the expression.

SimplifyQ[term2]

```
0
```

On the other hand, the standard Mathematica function “Simplify” does not simplify this function fully.

Simplify[term2]

$$\frac{1}{\sqrt{2}\,m\,\omega}\left(m\,\omega\sqrt{\frac{m\,\omega}{\hbar}} ** q[i] - i\,m\,\omega\frac{1}{\sqrt{\frac{m\,\omega}{\hbar}}\,\hbar} ** p[i] + \sqrt{\frac{m\,\omega}{\hbar}}\,(i\,p[i] - m\,\omega\,q[i])\right)$$

StandardOrderQ

StandardOrderQ will order the operators of an expression according to operator type, operator species, and site index respectively. Furthermore this function will place creation operators to the left of annihiliation operators of the same type, accounting for the commutator of the two operators.Operator product definitions are applied by default, and can be turned off by specifying ApplyDefinition -> False.Operators are sorted in the following order : {Bra, b†, b, f†, f, J, X, σ, p, q, Ket}. This function can be used to partially simplify functions but is not as extensive as the SimplifyQ function.

Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (q and p), Fermionic Creation and Annihilation operators (f and f†), and Bosonic Creation and Annihilation operators (b and b†).

term1 = q[j] ** p[i] ** f[k] ** f†[k] ** f[j] ** b[i] ** b†[i];

StandardOrderQ will reorder the operators in the expression above, accounting for the commutators of the two operators.

StandardOrderQ[term1]

$$(i\,\hbar\,\delta[j,i]) ** f[j] + f[j] ** p[i] ** q[j] + (i\,\hbar\,\delta[j,i]) ** n_b[i] ** f[j] + n_b[i] ** f[j] ** p[i] ** q[j] + 2\,(i\,\hbar\,\delta[j,i]) ** f^\dagger[k] ** f[j] ** f[k] + 2\,f^\dagger[k] ** f[j] ** f[k] ** p[i] ** q[j] + 2\,(i\,\hbar\,\delta[j,i]) ** n_b[i] ** f^\dagger[k] ** f[j] ** f[k] + 2\,n_b[i] ** f^\dagger[k] ** f[j] ** f[k] ** p[i] ** q[j]$$

TakeCPart

TakeCPart will scan input and return the “c” number terms (numbers and other constants). The output is returned as a list. Each entry in the list is the “c” number component of a single term found within the input expression, where terms are taken as components seperated by addition.

Example

In the DiracQ Palette activate only the Canonical Position and Momentum Operators (q and p). We have deliberately turned off the Boson operators b[i], so that they are part of the c-number answer. Turning them on as Bosons would group them with the q-number parts in correct order.

TakeCPart will yield the C numbers of the input.

```
TakeCPart[5 a b[i] ** p[i] ** θ[i] ** q[i]]
```

```
{5 a b[i] θ[i]}
```

If the input expression contains terms seperated by addition then TakeCPart will yield a list of C numbers of each term in the expression.

```
TakeCPart[5 a b[i] ** p[i] ** θ[i] ** q[i] + 3 j[i] ** p[i]]
```

```
{3 j[i], 5 a b[i] θ[i]}
```

TakeQPart

TakeQPart will scan input and return the Q number parts (operators). Output is returned as a list. Each entry in the list is the Q number component of a single term found within the input expression, where terms are taken as components seperated by addition.

Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (q and p).

TakeQPart will yield the Q numbers of the input.

```
TakeQPart[5 a b[i] ** p[i] ** θ[i] ** q[i]]
```

```
{p[i] ** q[i]}
```

If the input expression contains terms seperated by addition then TakeQPart will yield a list of Q numbers of each term in the expression.

```
TakeQPart[5 a b[i] ** p[i] ** θ[i] ** q[i] + 3 j[i] ** p[i]]
```

```
{p[i], p[i] ** q[i]}
```

TakeSummand

TakeSummand will return the summand of an input expression of the form Sum[Summand, Index (Indices)]. Input of other forms will yield error messages.

Example

In the DiracQ Palette activate Canonical Position and Momentum Operators (q and p).

TakeSummand will yield only the summand of a sum.

TakeSummand$\left[5 \sum_{i} p[i]\right]$

```
5 p[i]
```

```
TakeSummand[5 ∑ ∑ p[i] ** q[i] ** q[i]]
              i j
```

```
5 p[i] ** q[i] ** q[i]
```

Vacuum

Vacuum is the symbol used to represent the vacuum state. In general different operators are taken to act on different basis and therefore Vacuum represents the direct product of the vacuum state of several different basis.

▪ Example

In the DiracQ Palette activate Fermionic Creation and Annihilation operators (f and f†).

The Vacuum state can take the form of either a Bra or a Ket vector.

```
f[i] ⊗ Ket[Vacuum]
```

```
0
```

```
Bra[Vacuum] ⊗ f†[i]
```

```
0
```

Operators of any type can act on the vacuum state.

x

x is used to represent the x coordinate direction in several of the operators in the DiracQ package.

▪ Example

In the DiracQ Palette activate Pauli Spin Matrices (σ), Canonical Position and Momentum Operators (q and p), and Canonical Spin Matrices (J).

x is used by the Pauli Spin Matrices and the Canonical Spin Matrices.

```
Commutator[σ[i, x], σ[i, y]]
```

```
2 ⅈ σ[i, z]
```

```
Commutator[J[i, x], J[i, y]]
```

```
ⅈ ℏ ** J[i, z]
```

x is also used to represent coordinate direction of Position and Momentum operators.

```
p⃗[i]
```

```
{p[i, x], p[i, y], p[i, z]}
```

X

X is the Hubbard projection operator, three arguments are required for its definition. At a given site “i” the X[i,j,k] is the projection operator |j> < k|, where the ket |j> runs over the four possibilities of electron occupation at the given site “i”, namely |0> , |↑> , |↓> and |↑↓>. These four states are assigned the “j” values j=0,1,-1, 2 respectively. States with $j^2 + k^2$ = odd integer are Fermi like and with $j^2 + k^2$ = even integer are Boson like. We strongly recommend prescribing numerical values rather than symbolic values to the two indices “j” and “k” since the built in properties of (anti)-commutation are only valid in these cases.

Example

In the DiracQ Palette activate Hubbard Operators (X).

The Hubbard Operator is defined as follows : $X_i^{jk} = |j><k|$

```
ProductQ[X[i, a, d], X[i, d, c]]
```

```
X[i, a, c]
```

At a single site “i”, if required we can possibly override the recommendation of numerical values, and use symbolic values for the state labels.

```
Commutator[X[i, j, k], X[i, l, m]]
```

```
-δ[j, m] ** X[i, l, k] + δ[l, k] ** X[i, j, m]
```

```
AntiCommutator[X[i, a, d], X[i, d, c]]
```

```
δ[a, c] ** X[i, d, d] + X[i, a, c]
```

At different sites “i,j”, the use of numerical values is mandatory to get correct answers.

```
Commutator[X[i, 1, 1], X[j, 2, 0]]
```

```
0
```

```
AntiCommutator[X[i, 1, 1], X[j, 2, 0]]
```

```
2 X[i, 1, 1] ** X[j, 2, 0] - 2 δ[i, j] ** X[i, 1, 1] ** X[j, 2, 0]
```

```
Commutator[X[i, 1, 0], X[j, 2, -1]]
```

```
2 X[i, 1, 0] ** X[j, 2, -1] - 2 δ[i, j] ** X[i, 1, 0] ** X[j, 2, -1]
```

```
AntiCommutator[X[i, 1, 0], X[j, 2, -1]]
```

```
0
```

y

y is used to represent the y coordinate direction in several of the operators in the DiracQ package.

Example

In the DiracQ Palette activate Pauli Spin Matrices (σ), Canonical Position and Momentum Operators (q and p), and Canonical Spin Matrices (J).

y is used by the Pauli Spin Matrices and the Canonical Spin Matrices.

```
Commutator[σ[i, x], σ[i, y]]
```

```
2 ⅈ σ[i, z]
```

```
Commutator[J[i, x], J[i, y]]
```

```
ⅈ ℏ ** J[i, z]
```

x is also used to represent coordinate direction of Position and Momentum operators.

```
p⃗[i]
{p[i, x], p[i, y], p[i, z]}
```

z

z is used to represent the z coordinate direction in several of the operators in the DiracQ package.

▪ Example

In the DiracQ Palette activate Pauli Spin Matrices (σ), Canonical Position and Momentum Operators (q and p), and Canonical Spin Matrices (J).

z is used by the Pauli Spin Matrices and the Canonical Spin Matrices.

```
Commutator[σ[i, x], σ[i, y]]

2 ⅈ σ[i, z]

Commutator[J[i, x], J[i, y]]

ⅈ ℏ ** J[i, z]
```

x is also used to represent coordinate direction of Position and Momentum operators.

```
p⃗[i]
{p[i, x], p[i, y], p[i, z]}
```

δ

δ is the Kronecker delta. This function accepts two arguments and is equal to one if the arguments are equal, 0 if they are unequal, and will remain unevaluated if the arguments one or both of the arguments have not been assigned values. This function is similar to the standard *Mathematica* function KroneckerDelta but is different in some ways. If the option Evaluation is set to identical than the function will be equal to 0 unless the two arguments are identical.

▪ Example

In the DiracQ Palette activate Pauli Spin Matrices (σ).

```
δ[1, 1]

1

δ[1, 0]

0

δ[i, j]

δ[i, j]

δ[i, j, Evaluation → Identical]

0
```

SimplifyQ$\left[\sum_{i} \delta[i, j]\, i\right]$

```
j
```

ϵ

ϵ is the Levi - Civita completely antisymmetric symbol with three arguments. In this package the ϵ is only used for coordinate directions x, y, z. Any permutation of these symbols that follows from the right hand rule will yield one, any permutation opposite to the right hand rule yields - 1, and any argument that involves repeated symbols will yield zero.

Example

```
ϵ[x, y, z]
```

```
1
```

```
ϵ[x, z, y]
```

```
-1
```

```
ϵ[y, x, y]
```

```
0
```

σ

σ is the Pauli spin matrix. This operator requires two arguments. The first is site index and the second is coordinate direction. An optional third argument is used to denote different spin species. Also included are the Pauli raising and lowering operators, denoted by σ^{Plus} and σ^{Minus} respectively. The raising and lowering operators require only one argument corresponding to site. A second argument will be taken to represent spin species.

Example

In the DiracQ Palette activate Pauli Spin Matrices (σ).

The Pauli Spin Matrices obey the familiar algebraic relations.

```
Commutator[σ[i, x], σ[i, y]]
```

```
2 ⅈ σ[i, z]
```

```
σ[i, x] ⊗ σ[i, x]
```

```
1
```

The Pauli raising and lowering operators are included as secondary operators

σ^{Plus}`[i]`

```
σ[i, x] + ⅈ σ[i, y]
```

σ^{Minus}`[j]`

```
σ[j, x] - ⅈ σ[j, y]
```

The optional third argument is used to denote different spin species, and therefore two operators of different species will commute.

```
Commutator[σ[i, x, 1], σ[i, y, 2]]
```

```
0
```

†

The dagger symbol, †, is used in the representation of creation operators. The dagger symbol is not used as a superscript but is rather placed directly following the symbol ' f' for fermionic operators and ' b' for bosonic operators. To enter the dagger symbol find the symbol in the DiracQ palette or one of the other *Mathematica* palettes or enter ⎋dg⎋.

ℏ

ℏ is the reduced Planck' s constant. To enter ℏ find the symbol in the DiracQ palette or one of the other *Mathematica* palettes or enter ⎋hb⎋.