4.4.4 Time integral operator

The **idt** operator computes the time-integral of its argument, as shown in Table 4-13.

Table 4-13—Time integral

Operator	Comments
idt(*expr*)	Returns $\int_{t_0}^{t} x(\tau)d\tau + c$, where $x(\tau)$ is the value of *expr* at time τ , t_0 is the start time of the simulation, t is the current time, and c is the initial starting point as determined by the simulator and is generally the DC value (the value that makes *expr* equal to zero)
idt(*expr,ic*)	Returns $\int_{t_0}^{t} x(\tau)d\tau + c$, where in this case c is the value of *ic* at t_0 .
idt(*expr,ic,assert*)	Returns $\int_{t_a}^{t} x(\tau)d\tau + c$, where c is the value of *ic* at t_a , which is the time when *assert* was last nonzero or t_0 if *assert* was never nonzero.
idt(*expr,ic,assert,abstol*)	Same as above, except the absolute tolerance used to control the error in the numerical integration process is specified explicitly with *abstol*.
idt(*expr,ic,assert,nature*)	Same as above, except the absolute tolerance used to control the error in the numerical integration process is take from the specified nature.

When specified with initial conditions, **idt()** returns the value of the initial condition in DC or IC analyses. If *assert* is specified and nonzero, **idt()** will return the value of *ic* at the time when *assert* was last nonzero. If *assert* was never nonzero, **idt()** will return the DC / IC (time zero) value of *ic*. Without initial conditions, **idt()** multiplies its argument by infinity in DC analysis. Hence, without initial conditions, it can only be used in a system with feedback which forces its argument to zero (0).

The optional parameter *abstol* or *nature* is used to derive an absolute tolerance if needed. Whether an absolute tolerance is needed depends on the context where **idt()** is used. (See Section 4.4.2 for more information.) The absolute tolerance applies to the input of the **idt** operator and is the largest signal level that is considered negligible.

4.4.5 Circular integrator operator

The **idtmod** operator, also called the *circular integrator*, converts an expression argument into its indefinitely integrated form similar to the **idt** operator, as shown in Table 4-14.

Table 4-14—Circular integrator

Operator	Comments
idtmod(*expr*)	Returns $\int_{t_0}^{t} x(\tau)d\tau + c$, where $x(\tau)$ is the value of *expr* at time τ , t_0 is the start time of the simulation, t is the current time, and c is the initial starting point as determined by the simulator and is generally the DC value (the value that makes *expr* equal to zero)

Table 4-14—Circular integrator

Operator	Comments
idtmod(*expr,ic*)	Returns $\int_{t_0}^{t} x(\tau)d\tau + c$, where in this case c is the value of *ic* at t_0 .
idtmod(*expr,ic,modulus*)	Returns *k*, where $0 \le k <$ `modulus` and *k* is $\int_0^t x(\tau)d\tau + c = n \times modulus + k$, *n* = ... -3,-2,-1,0,1,2,3.... Where c is the value of *ic* at t_0 .
idtmod(*expr,ic,modulus,offset*)	Returns *k*, where `offset` $\le k <$ `offset` + `modulus` and *k* is $\int_0^t x(\tau)d\tau + ic = n \times modulus + k$ Where c is the value of *ic* at t_0 .
idtmod(*expr,ic,modulus,offset, abstol*)	Same as above, except the absolute tolerance used to control the error in the numerical integration process is specified explicitly with *abstol*.
idtmod(*expr,ic,modulus,offset, nature*)	Same as above, except the absolute tolerance used to control the error in the numerical integration process is take from the specified nature.

The initial condition is optional. If the initial condition is not specified, it defaults to zero (`0`). Regardless, the initial condition shall force the DC solution to the system.

If **idtmod()** is used in a system with feedback configuration which forces `expr` to zero (`0`), the initial condition can be omitted without any unexpected behavior during simulation. For example, an operational amplifier alone needs an initial condition, but the same amplifier with the right external feedback circuitry does not need a forced DC solution.

The output of the **idtmod()** function shall remain in the range

```
offset <= idtmod < offset+modulus
```

The `modulus` shall be an expression which evaluates to a positive value. If the modulus is not specified, then **idtmod()** shall behave like **idt()** and not limit the output of the integrator.

The default for `offset` shall be zero (`0`).

The following relationship between **idt()** and **idtmod()** shall hold at all times.

Examples:

If

```
y = idt(expr, ic) ;
z = idtmod(expr, ic, modulus, offset) ;
```

then

```
y = n * modulus + z ;// n is an integer
```

where

```
offset ≤ z < modulus + offset
```

In this example, the circular integrator is useful in cases where the integral can get very large, such as a VCO. In a VCO we are only interested in the output values in the range [0,2π], e.g.,

```
phase = idtmod(fc + gain*V(in), 0, 1, 0);
V(OUT) <+ sin(2*`M_PI*phase);
```

Here, the circular integrator returns a value in the range [0,1].

4.4.6 Derivative operator

ddx() provides access to symbolically-computed partial derivatives of expressions in the analog block. The analog simulator computes symbolic derivatives of expressions used in contribution statements in order to use Newton-Raphson iteration to solve the system of equations. In many cases in compact modeling, the values of these derivatives are useful quantities for design, such as the trans conductance of a transistor (gm) or the capacitance of a nonlinear charge-storage element such as a varactor. The syntax for this operator is shown in Syntax 4-2.

ddx_call ::=
ddx (analog_expression, branch_probe_function_call **)**

Syntax 4-2—Syntax for the derivative operator

The operator returns the partial derivative of its first argument with respect to the unknown indicated by the second argument, holding all other unknowns fixed and evaluated at the current operating point. The second argument shall be the potential of a scalar net or port or the flow through a branch, because these are the unknown variables in the system of equations for the analog solver. For the modified nodal analysis used in most SPICE-like simulators, these unknowns are the node voltages and certain branch currents.

If the expression does not depend explicitly on the unknown, then **ddx()** returns zero (0). Care must be taken when using implicit equations or indirect assignments, for which the simulator may create internal unknowns; derivatives with respect to these internal unknowns cannot be accessed with **ddx()**.

Unlike the **ddt()** operator, no tolerance is required because the partial derivative is computed symbolically and evaluated at the current operating point.

Examples:

This first example uses **ddx()** to obtain the conductance of the diode. The variable `gdio` is declared as an output variable (see Section 3.1.1) so that its value is available for inspection by the designer.