$ K # + Expression Reference

Syntax

This section describes the basic syntax of an expression, describing how it is built up from factors, terms and simple expressions.

Operators

There are four basic types of operators used in expressions. They are:
Arithmetic operators
Logical operators
String operators
Relational operators

Function Calls

The general syntax of function calls is described in this section.

Standard Functions

There is a library of standard functions which can be called in expressions. These fall into three categories.
Arithmetic Functions
String Functions
Conditional Expressions

$ Expression Reference
K Reference, Expression
# EXPRESSION_REFERENCE
+ EX:00

$ K + Expression syntax

Expressions are made up of operators and operands. Most operators are binary; they take two operands. The rest are unary and take only one operand. Binary operators use the usual algebraic form (for example, A + B). A unary operator always precedes its operand (for example, -B). In more complex expressions, rules of precedence clarify the order in which operations are performed.

Operators	**Precedence**	**Categories**
not	first (high)	unary operators
^	second	exponent operator
*, /, div, mod, and, shl, shr, as	third	multiplying operators
+,-, or, xor	forth	adding operators
=, <>, <, >, <=, >=	fifth	relational operators

There are three basic rules of precedence:

An operand between two operators of different precedence is bound to the operator with higher precedence.

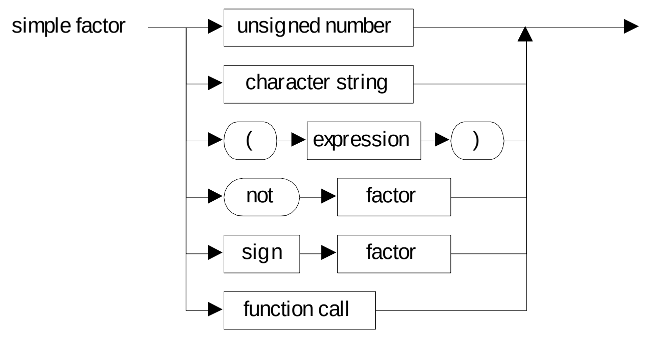
An operand between two equal operators is bound to the one on its left.

Expressions within parentheses are evaluated prior to being treated as a single operand.

Operations with equal precedence are normally performed from left to right.

The precedence rules follow from the syntax of expressions, which are built from simple factors, factors, terms, and simple expressions.

A Simple factor's syntax follows:



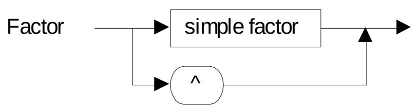
A function call activates a function and denotes the value returned by the function.

These are some examples of simple factors:

```
15                  { Unsigned constant }
(X + Y + Z)         { Subexpression }
Sin(X / 2)          { Function call }
not Done            { Negation of a Boolean }
```

Factors apply the exponent operator to simple factors:

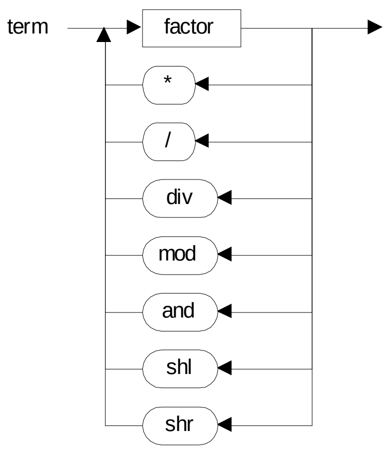
# EXP_SYNTAX
$ Expression Syntax
K Syntax, expressions
+ EX:10



These are some examples of factors:

```
2^3
pi^2
```

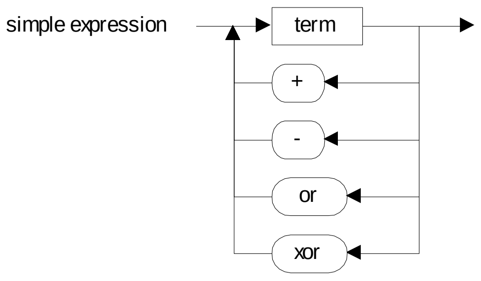
Terms apply the multiplying operators to factors:



Here are some examples of terms:

```
X * Y
Z / (1 - Z)
Y shl 2
(X <= Y) and (Y < Z)
```

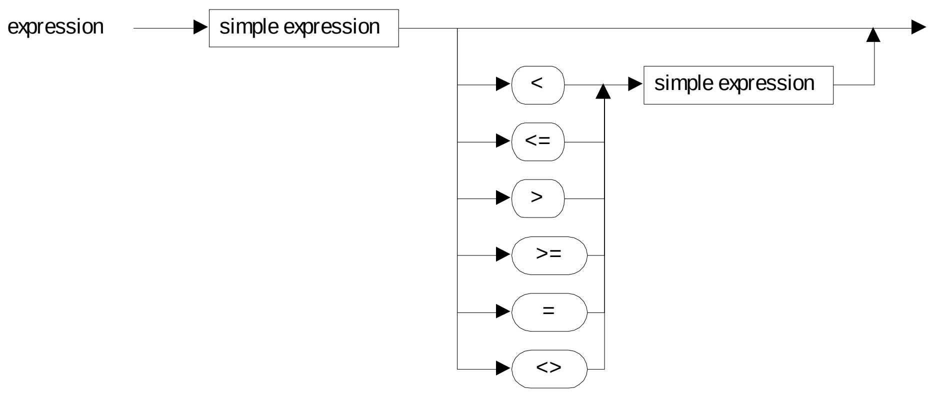
Simple expressions apply adding operators and signs to terms:



Here are some examples of simple expressions:

```
X + Y
-X
Hue1 + Hue2
I * J + 1
```

An expression applies the relational operators to simple expressions:



Here are some examples of expressions:

```
X = 1.5
Done <> Error
(I < J) = (J < K)
C in Hue1
```

$ K + Arithmetic operators

The following tables show the types of operands and results for binary and unary arithmetic operations.

Binary arithmetic operators

Operator	**Operation**	**Operand types**	**Result type**
^	exponent	integer type	*float*
		float type	*float*
+	addition	integer type	*integer*
		float type	*float*
-	subtraction	integer type	*integer*
		float type	*float*
*	multiplication	integer type	*integer*
		float type	*float*
/	division	integer type	*float*
		float type	*float*
div	integer division	integer type	*integer*
mod	remainder	integer type	*integer*

The + operator is also used as a string operator
.

Unary arithmetic operators

Operator	**Operation**	**Operand types**	**Result type**
+	sign identity	integer type	*integer*
		float type	*float*
-	sign negation	integer type	*integer*
		float type	*float*

If both operands of a +, -,*, div, or mod operator are of an integer type, the result type is of the common type of the two operands.

If one or both operands of a +, -, or * operator are of a float type, the type of the result is *Float*.

If the operand of the sign identity or sign negation operator is of an integer type, the result is of the same integer type. If the operator is of a float type, the type of the result is *Float*.

The value of *X* / *Y* is always of type *Float* regardless of the operand types. An error occurs if *Y* is zero.

The value of *I* div *J* is the mathematical quotient of *I* / *J*, rounded in the direction of zero to an integer value. An error occurs if *J* is zero.

The mod operator returns the remainder obtained by dividing its two operands; that is,
mod J = I - (I div J) * J
The sign of the result of mod is the same as the sign of *I*. An error occurs if *J* is zero.

# EXP_OP_ARITHMETIC
$ Arithmetic operators
K Operators, arithmetic
+ EX:20

[#] [$] [K] [+] Logical Operators

There are two classes of logical operators. Bitwise, which operate on integers, and Boolean which operate on Boolean types only.

Bitwise logical operators
The types of operands and results for logical operations are shown in the following table.

Operator	**Operation**	**Operand types**	**Result type**
not	bitwise negation	integer type	*Integer*
and	bitwise and	integer type	*Integer*
or	bitwise or	integer type	*Integer*
xor	bitwise xor	integer type	*Integer*
shl	shift left	integer type	*Integer*
shr	shift right	integer type	*Integer*

If the operand of the not operator is of an integer type, the result is of the same integer type. The not operator is a unary operator.

If both operands of an and, or, or xor operator are of an integer type, the result type is the common type of the two operands.

The operations *I shl J* and Ishr *J* shift the value of *I* to the left right by *J* bits. The result type is the same as the type of *I*.

Boolean logical operators
The types of operands and results for Boolean operations are shown in the following table.

Operator	**Operation**	**Operand types**	**Result type**
not	negation	Boolean type	*Boolean*
and	logical and	Boolean type	*Boolean*
or	logicalor	Boolean type	*Boolean*
xor	logical xor	Boolean type	*Boolean*

Normal Boolean logic governs the results of these operations. For instance, *A* and *B* is *True* only if both *A* and *B* are *True*.

[#] EXP_OP_LOGICAL
[$] Logical operators
[K] Bitwise operators; Boolean operators; Operators, logical; Operators, bitwise; Operators, boolean
[+] EX:30

[#] [$] [K] [+] String operators

The types of operands and results for string operation are shown in the following table.

Operator	Operation	Operand types	Result type
+	concatenation	string type	*String*

The result of the operation *S* + *T*, where S and T are of a string type is the concatenation of *S* and *T*.

[#] EXP_OP_STRING
[$] String operators
[K] Operators, string; Concatenate; Strings, combining
[+] EX:40

[#] [$] [K] [+] Relational operators

The types of operands and results for relational operations are shown in the following table.

Operator	Operation	Operand types	Result type
=	equal	compatible	*Boolean*
<>	not equal	compatible	*Boolean*
<	less than	compatible	*Boolean*
>	greater than	compatible	*Boolean*
<=	less than or equal to	compatible	*Boolean*
>=	greater than or equal to	compatible	*Boolean*

Comparing simple types
When the operands =, <>, <, >, >=, or <= are of simple types, they must be compatible types; however, if one operand is of a float type, the other can be of an integer type.

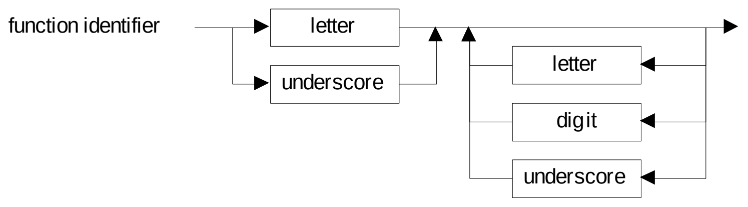
Comparing strings
The relational operators =, <>, <, >>, >=, and <= compare strings according to the ordering of the extended ASCII character set. Any two string values can be compared because all string values are compatible.

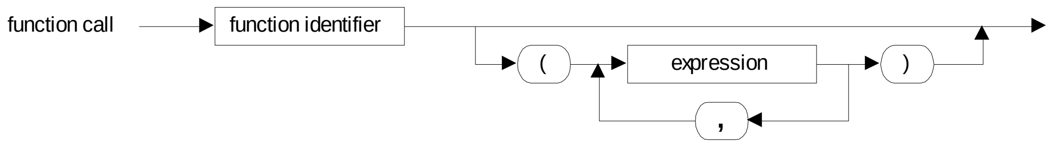
[#] EXP_OP_RELATIONAL
[$] Relational operators
[K] Operators, relational; Less than; Greater than; Equal to; Not Equal to
[+] EX:50

[#] [$] [K] [+] Function calls

A function call activates a function specified by a function identifier. The function call must have a list of actual parameters if the corresponding function declaration contains a list of formal parameters. Each parameter takes the place of the corresponding formal parameter.

A function identifier is a character string made up of letters, digits and the underscore character ('_'). A valid identifier may not start with a digit.





These are some examples of function calls:

```
arctan(1.572)
pi
pos('a', 'concatenate')
```

[#] EXP_FUNCTION
[$] Function calls
[K] Function calls, syntax; Syntax, of function calls; Function identifiers; Identifiers
[+] EX:60

[#] [$] [K] [+] Arithmetic functions

function Trunc(X: Extended):Integer
The Trunc function truncates a float-type value to an integer-type value. X is a float-type expression. Trunc returns an Integer value that is the value of X rounded toward zero.

function Round(X: Extended): Integer
The Round function rounds a float-type value to an integer-type value. X is a float-type expression. Round returns a Longint value that is the value of X rounded to the nearest whole number. If X is exactly halfway between two whole numbers, the result is the number with the greatest absolute magnitude.

function Abs(X): Float
The Abs function returns the absolute value of the argument. X is an integer-type or float-type expression.

function ArcTan(X: Float): Float
ArcTan calculates the arctangent of the given number. Calculate other trigonometric functions using Sin, Cos, and ArcTan in the following expressions
Tan(x) = Sin(x) / Cos(x)
ArcSin(x) = ArcTan (x/sqrt (1-sqr (x)))
ArcCos(x) = ArcTan (sqrt (1-sqr (x)) /x)

function Cos(X: Float): Float
The Cos function returns the cosine of the angle X, in radians.

function Exp(X: Float): Float
Exp returns the value of e raised to the power of X, where e is the base of the natural logarithms.

function Frac(X: Float): Float
The Frac function returns the fractional part of the argument X. X is a float-type expression. The result is the fractional part of X; that is: **Frac(X) = X - Int(X)**.

function Int(X: Float): Float
X is a float-type expression. The result is the integer part of X; that is, X rounded toward zero.

function Ln(X: Float): Float
The Ln function returns the natural logarithm (Ln(e) = 1) of the float-type expression X.

function Pi: Float
Use Pi in mathematical calculations that require pi, the ratio of a circle's circumference to its diameter. Pi is approximated as 3.1415926535897932385.

function Sin(X: Float): Float
The Sin function returns the sine of the argument. X is a float-type expression. Sin returns the sine of the angle X in radians.

function Sqr(X: Float): Float
The Sqr function returns the square of the argument. X is a floating-point expression. The result, of the same type as X, is the square of X, or X*X.

function Sqrt(X: Float): Float
X is a floating-point expression. The result is the square root of X.

function Power(Base, Exponent: Float): Float

[#] EXP_STD_ARITHMETIC
[$] Arithmetic functions
[K] Functions, arithmetic; functions, trig; Sin; Cos; ArcTan; Trunc; Round; Abs; Exp; Ln; Sqrt; Sqr; Frac; Int
[+] EX:70

The Power function raises Base to any power. For fractional exponents or exponents greater than MaxInt, Base must be greater than 0.

[#] [$] [K] [+] String Functions

function Upper(S: string): string
The Upper function returns a string containing the same text as S, but with all 7-bit ASCII characters between 'a' and 'z' converted to uppercase.

function Lower(S: string): string
Lower returns a string with the same text as the string passed in S, but with all letters converted to lowercase. The conversion affects only 7-bit ASCII characters between 'A' and 'Z'.

function Copy(S: string; Index, Count: Integer): string
The Copy function returns a substring of a string. S is a string-type expression. Index and Count are integer-type expressions. Copy returns a string containing Count characters starting at S[Index]. If Index is larger than the length of S, Copy returns an empty string. If Count specifies more characters than are available, the only the characters from S[Index] to the end of S are returned.

function Pos(Substr: string; S: string): Integer
Pos searches for a substring, Substr, in a string, S. Substr and S are string-type expressions. Pos searches for Substr within S and returns an integer value that is the index of the first character of Substr within S. Pos ignores case-insensitive matches. If Substr is not found, Pos returns zero.

function Length(S: string): Integer
The Length function returns the number of characters actually used in the string S.

[#] EXP_STD_STRING
[$] String functions
[K] Functions, string; Upper(case); Lower(case); Copy (String function); Pos (String function); Length, of string; Case of string, changing; Substring, extracting; Substring, finding
[+] EX:80

[#] [$] [K] [+] Conditional Expression

function If(Condition: Boolean, TrueResult, FalseResult): ResultType
Condition is a Boolean expression. When the function is evaluated, it returns TrueResult if Condition else FalseResult. TrueResult and FalseResult need not be of the same type and result type of the IF expression may change depending on Condition.

[#] EXP_STD_CONDITIONAL
[$] Conditional expression
[K] Expressions, conditional, IF (Conditional expression)
[+] EX:90