K

Reserved words, special symbols, operator precedence

K.1 OVERVIEW

This chapter lists the reserved words — including keywords for the external interface sublanguages —, the reserved special (non-alphabetic) symbols, and the precedence of operators appearing in expressions.

K.2 RESERVED WORDS

Following are the sixty-two reserved words of Eiffel, in alphabetical order.

Recall the distinction between *reserved words* and their special case, *keywords*. Reserved words include all the names (listed below) that cannot be used as identifiers for classes, features or entities. Some reserved words carry a meaning of their own, such as *Current* which denotes an expression and *TUPLE* which denotes a type. These are typeset in italics, with a first letter in upper case (all letters upper-case in the case of a type or class name). Reserved words that do *not* by themselves denote anything but just serve as syntactic markers, such as **do** or **if**, are called keywords and appear in boldface.

Every reserved word (keyword or not) has an entry in the index, with a reference to the page of the corresponding syntax productions, if any.

agent	**alias**	**all**	**and**	**as**	**assign**	**attribute**
check	**class**	**convert**	**create**	*Current*	**debug**	**deferred**
do	**else**	**elseif**	**end**	**ensure**	**expanded**	**export**
external	*False*	**feature**	**from**	**frozen**	**if**	**implies**
inherit	**inspect**	**invariant**	**like**	**local**	**loop**	**not**
note	**obsolete**	**old**	**once**	**only**	**or**	*Precursor*
redefine	**rename**	**require**	**rescue**	*Result*	**retry**	**select**
separate	**then**	*True*	*TUPLE*	**undefine**	**until**	**variant**
Void	**when**	**xor**				

K.3 SPECIAL SYMBOLS

The following table shows all the special symbols of the language, together with the page of the syntax productions where they appear.

← This table appeared first on page 880.

Symbol	**Name**	**Role**	**Pages**
--	Double dash	Introduces comments.	
;	Semicolon	Separates instructions, declarations, assertion clauses…; always optional.	
,	Comma	Separates elements in lists of of entities or expressions.	
:	Colon	Separates the Type_mark in a declaration, a Tag_mark in an Assertion_clause, and a Note_name term in a Notes clause.	
:? :!	Colon-question, colon-exclamation	Separate the Type_mark in a declaration.	
'	Single quote	Encloses manifest constants.	
"	Double quote	Encloses manifest strings.	
%	Percent	Introduces special character codes.	
/	Slash	In a special character code, introduces a character through its code.	
+ –	Plus and minus	Signs of integer and real constants. (Also permitted as prefix and infix operators, appearing in a separate table.)	
$	Dollar	Address operator for passing the address of an Eiffel feature or expression to a routine (usually external).	
%	Percent	Introduces a special character code.	
/	Slash	In a special character, introduces a character by its numerical code.	
.	Dot	Separates target from feature in a feature call or creation call. Separates integer from fractional part in a real number.	
–>	Arrow	Introduces the constraint of a constrained formal generic parameter.	
:=	Receives	Assignment operator.	
= /=	Equal, not-equal signs	Equality and non-equality operators.	
~ /~	Tilde, slash-tilde	Object equality and non-equality operators.	
( )	Parentheses	Group subexpressions in operator expressions; enclose formal and actual arguments of routines.	
(| |)	Target parentheses	Enclose a constant or non-atomic expression used as target of a call in dot or bracked notation.	
[ ]	Brackets	Enclose formal and actual generic parameters to classes; enclose items of a manifest tuple; specify that a feature has a Bracket alias.	
{ }	Braces	Enclose types in various contexts: Clients part, Feature_clause or New_export_list, Creation_type.	

K.4 OPERATORS AND THEIR PRECEDENCE

Operator precedence levels

13 **.** (Dot notation, in qualified and non-object calls)

12 **old** (In postconditions)
not + **–Used as unary**
All free unary operators

11 All free binary operators.

10 ^ (Used as binary: power)

9 ∗ / // \\ (As binary: multiplicative arithmetic operators)

8 **+ – Used as binary**

7 **. .** (To define an interval)

6 = /= ~ /~ < > <= >= (As binary: relational operators)

5 **and** **and then**
(Conjunctive boolean operators)

4 **or** **or else** **xor**
(Disjunctive boolean operators)

3 **implies**(Implicative boolean operator)

2 **[ ]**(Manifest tuple delimiter)

1 **;** (Optional semicolon between
an Assertion_clause and the next)

K.5 KEYWORDS AND SYMBOLS OF SPECIAL INTERFACE SUBLANGUAGES

Here are the keywords used inthe special interface sublanguages. These are not Eiffel keywords, but special words that may appear in strings denoting external languages and their special mechanisms.

C	**C++**	**data_member**	**delete**
Fortran95	**include**	**inline**	**Java**
macro	**new**	**static**	**struct**

The following symbols may appear in such strings:

Symbol	**Name**	**Role**
:	Colon	Introduces the result type in a function signature.
( )	Parentheses	Enclose argument types in a function signature.
"	Double quote	Encloses a file name (may have to be written **%"** as part of a manifest string).
$	Dollar	Introduces an Eiffel entity in an inline C text.

< >	Angle brackets	Enclose the name of a system include file.
[ ]	Square brackets	Enclose macro and DLL specifications.