Language changes from the previous edition

F.1 OVERVIEW

Stability has been the principal characteristic of Eiffel's history since the language was designed on 27 September 1985. The concepts behind the language, the structure of software texts, and the principal constructs have remained the same. There have of course been significant changes:

- ISE Eiffel 2.1 (1988) introduced constrained genericity and the Assignment Attempt mechanism.
- Versions 2.1 to 2.3 introduced expanded types, double-precision reals, expanded classes and types, the join mechanism for deferred features, assignment attempt, the Indexing clause, infix and prefix operators, the Obsolete clause, Unique values, the Multi_branch instruction.
- The transition from Eiffel 2 to Eiffel 3 (1990-1993) was the opportunity for a general cleanup of the language, unification and simplification of the concepts; in particular it made basic types full-fledged classes, to yield a completely consistent type system, and got rid of special features such as *Forget*, so that feature call always applies to objects rather than references. The first edition of this book officially introduced Eiffel 3; by providing the complete reference for a full-function language, it permitted the growth of the Eiffel industry and served as the basis for all current commercial and non-commercial compilers.
- Eiffel 4 (in particular ISE's Eiffel 4.2 in 1998 and 4.3 to 4.5 in 1999) introduced the Precursor construct, recursive generic constraints, tuples, agents, creation expressions and a new creation syntax.
- The present edition describes Eiffel 5, which brings a few significant improvements, although it remains close to previous versions. In the Eiffel tradition, the changes are not so much extensions (we are constantly wary of the danger of "creeping featurism") as efforts to make the language cleaner, simpler, more consistent, easier to learn, easier to use. This revision also *removed* a number of mechanisms (*BIT* types, Strip expressions), for which better alternatives were found to be available.

This appendix describes the language changes from the preceding edition to the present one, which are also the changes from Eiffel 3 to Eiffel 5.

Since the majority of Eiffel 5 users with pre-Eiffel-5 experience started with Eiffel 3, the pre-Eiffel-3 changes are of mostly historical interest. For that reason they appear in a separate appendix.

After a note on compatibility, the presentation of Eiffel 5 changes will successively consider: new constructs and semantic changes; lexical and syntactic changes; changes to validity constraints and conformance rules.

F.2 UPWARD COMPATIBILITY

The transition from Eiffel 2 to Eiffel 3 required changing some ways of expressing fundamental operations, such as comparison to *Void*. Accordingly, a translator was made available by ISE at the time.

The changes from Eiffel 3 to Eiffel 5 may only cause minor incompatibilities for existing Eiffel 3 software:

- The following new reserved words may not be used as identifiers: **assign**, **create**, *Precursor*, **pure**, **reference**, *TUPLE*.

  The keywords **creation** and **select** have been removed but compilers may continue to support them for a while, so you should refrain from using them as identifiers.

- If you had a feature called *default_create*, you should find another name, unless you wish to use it as a redefinition of the corresponding feature from *GENERAL*.
- If you had classes called *FUNCTION*, *PROCEDURE*, *ROUTINE* or *TYPE*, they will conflict with the corresponding new classes from the Kernel Library, so you should use a different name.
- In an Indexing clause the initial colon-terminated Index term, previously optional, is now required; you will have to add it if missing.
- Creation is now written **create** *x* rather than **!!** *x* and **create {***TYPE***}** *x* rather than **!** *TYPE* **!** *x*. This is the most visible syntax change, but does not raise any immediate concern since compilers should continue to support the previous syntax for several years. (This is the case with ISE Eiffel.) A translator does not appear necessary, although some scripts may be made available to update creation instructions to the new form.

Incompabilities may also result from the removal of *BIT* types and Strip expressions. The new bit manipulation features of class *INTEGER* provide a superior replacement for *BIT* types; Strip expressions were rarely used and their effect can be obtained in a simpler way through the agent mechanism. Here too compilers such as ISE Eiffel will continue to support the older mechanisms for several years.

F.3 NEW CONSTRUCTS AND SEMANTIC CHANGES

The generic mechanism now explicitly supports "recursive generic constraints", in which a constraint for a generic parameter may involve another (or the same) generic parameter, as in **class** *C* [*G*, *H* –> *ARRAY* [*G*]].

As noted in the preceding section, a class may now be declared as deferred even if it has no deferred feature. This makes it non-instantiable like any other deferred class.

Tuples (anonymous classes) are new. *Chapter 13.*

The *agent* mechanism (using tuples) is new. *Chapter 25.*

The semantics of *creation* has been made simpler, for creation instructions that do not explicitly list a creation procedure, by assuming that this uses the *default_create* procedure, introduced in *GENERAL* and redefinable in any class. *Chapter 20.*

The *generic creation* mechanism, making it possible to create objects of a Formal_generic_name type, is new. *Chapters 12 and 20.*

Creation expressions are new. (Pre-Eiffel-5, only creation instructions were available.) *20.14, page 441.*

The anchor of an Anchored type **like** *anchor* may now itself be anchored, as long as there is no cycle in the anchoring structure. In addition it is now possible to use an expanded or formal generic anchor. With the exception of expanded anchors this officializes possibilities that ISE Eiffel has supported for a long time. *11.14, page 247.*

The Precursor construct is new, replacing techniques (still applicable in complex cases) relying on repeated inheritance. *10.22, page 211.*

The notion of *pure* routine, useful to avoid unwanted side effects especially in assertions and concurrent computation, is new. *Chapters 8 and 20.*

The *feature name consistency principle* is new, at least in its full generality. The difference between operator and identifier features has always been one of syntax only; what is new is the possibility to use an operator feature as a normal feature name in dot notation, as in *a*.**infix** "+" (*b*). This convention is for generality and consistency rather than to address an acute practical need. *Page 87.*

The *once routine* mechanism has gained new flexibility through the introduction of "once keys" allowing "once per thread", "once per object", and manual control through the new class *ONCE_MANAGER*. *"ONCE ROUTINES", 23.11, page 508.*

Verbatim strings are new. *.27.7, page 617.*

Multi-branch instructions support two new forms, one discriminating on strings (in addition to the integers and characters previously supported), the other on the type of an object. *"MULTI-BRANCH CHOICE", 16.5, page 358.*

Unique constants are now guaranteed always to have different values even if they are declared in different classes. This has also led to the loosening of a validity constraint on Multi_branch instruction.

.“Unique Declaration rule”, page 396; comment on constraint COMB (page 363) ond page 899 below.

The arithmetic types have been developed and made more precise. The sized variants *INTEGER_8*, *INTEGER_16* and *INTEGER_64* are new. Also new is the explicit specification that *INTEGER* represents 32-bit integers, *REAL* 32-bit reals, and *DOUBLE* 64-bit reals.

A new *conversion mechanism* generalizes the ad hoc conformance rules that allowed conformance of *INTEGER* to *REAL* and of *INTEGER* and *REAL* to *DOUBLE*, as well as the “balancing rule” which permitted mixed-mode arithmetic, as in *your_integer + your_double*. Instead, there is now a general-purpose conversion and expression balancing mechanism, used by the basic types in the Kernel Library but applicable to any other classes.

“CONVERTIBILITY”, 14.13, page 301; “TARGET CONVERSION FOR INFIX FUNCTIONS”, 26.14, page 608.

Equality semantics now specifies that two objects cannot be equal unless their types are identical; previously, it was possible for an object to be equal to one of conforming type. The main reason for this change was to follow mathematical tradition by ensuring that equality is fully symmetric. Correspondingly, *copy semantics* requires an argument of type is identical — not just conforming — to the type of the target.

“OBJECT EQUALITY”, 21.5, page 457, and “COPYING AN OBJECT”, 21.2, page 447.

The *global inheritance structure* has been simplified: *ANY* no longer has ancestors *GENERAL* and *PLATFORM*. *GENERAL* has gone away altogether, so that *ANY*’s features are declared in *ANY* itself. *PLATFORM* is still there, but as a supplier rather than ancestor of *ANY*, through a new query *platform* of type *PLATFORM* in *ANY*, providing access to platform-specific properties.

Appendix A.

The notion of Class_type_reference, of the form **reference** *T*, is new. It has made it possible to simplify the Kernel Library by removing classes such as *INTEGER_REF* (see below).

11.12, page 245.

Non-conforming inheritance was present in the case of inheritance from an expanded class, but has been generalized to permit a Parent clause of the form **inherit expanded** *C*, hereby providing a simpler solution to the issues of repeated inheritance and removing the need for Select.

“NON-CONFORMING INHERITANCE”, 6.9, page 108.; “THE CASE OF REDECLARED FEATURES”, 15.5, page 321

Although *external features* have always been present, they originally supported only a Language_name, such as "C", and an optional **alias** specification (External_name). The inclusion of mini-sublanguages allowing detailed C specifications comes from ISE Eiffel 3, which provided direct support for C macros, include files and DLLs. Changes from that version include: removing of 16-bit DLL support (technically obsolete); replacing the keyword **dll32** and the class name *DLL_16* by **dll** and *DLL*; accepting routine names as well as routine indexes in **dll** specifications; specifying that in the absence of an **alias** subclause the name to be passed to the external language is the lower name of the external

Chapter 28.

Eiffel feature ; replacing the vertical bar |, used to introduce include files, by the keyword **include**. ISE Eiffel 4 introduced C++-specific mechanisms, allowing an Eiffel class to use the member functions, static functions, data members, constructors and destructors of a C++ class. That version also introduced the Legacy++ class wrapper and the Java interface. Eiffel 5 adds support for **inline** C functions and C **struct** specifications. The Cecil library mechanisms have also been considerably refined and extended based on extensive experience with the library.

In an Address argument, denoting the address of a value to be passed to external software, it is now permitted to use a Parenthesized expression , as in *external_routine* (**$** (*a* + *b*)), or a constant feature (previously excluded by clause 4 of the Argument validity rule). What is passed to the external routine is the address of a location containing the expression or constant.

"PASSING THE ADDRESS OF AN EIFFEL FEATURE", 28.8, page 646.

F.4 KERNEL LIBRARY CHANGES

A number of changes have been brought to the Kernel Library (ELKS); only the most important ones will be listed here.

Appendix A.

The names of features for comparison, object duplication and copying have been made more consistent, as shown by the following tables. Asterisks indicate new names — for existing features or, in the case of *twin*, new ones; names in roman and in parentheses indicate previous names.

OBJECT EQUALITY	Between arguments	Between target and argument
Redefinable	*equal* **infix** "}={" <—	*is_equal*
Frozen	**identical* <— (standard_equal)	**is_identical* <— (standard_is_equal)

OBJECT DUPLICATION	Of argument	Of target
Redefinable	*clone*	*twin*
Frozen	**identical_clone* (standard_clone)	**identical_twin*

OBJECT COPY	Of argument onto target
Redefinable	*copy*
Frozen	*identical_copy* <— (standard_copy)

The purpose of this change is to make the names uniform and easy to remember:

- Add *is_*for queries applying to the target: *equal* (*x*, *y*) compares its arguments, *x*.*is_equal* (*y*) compares the argument to the target.
- Use *identical* for frozen (non-redefinable) operations, which guarantee the original semantics of field-by-field equality or copying: *equal* and *copy* are redefinable, *identical* and *identical_copy* are not. Note that *clone* and its target-oriented variant *twin* are not directly redefinable, but they follow the redefinitions of *copy*.

The previous conventions were not bad, but the new ones seem a little better, especially with the introduction of twin.

infix "}={" is a new synonym of *equal*, making it a little easier to express object equality as *a* }={ *b*. (The symbol suggests an equal sign opening up both left and right to embrace the objects denoted by the operands.)

In addition, as noted in the previous section, copy and equality features now use type identity rather than type conformance between their arguments. This has led to a stronger precondition for *copy*, using *same_type* rather than *conforms_to*.

Thanks to the introduction of Class_type_reference, it has been possible to remove classes *INTEGER_REF*, *CHARACTER_REF* and so on; the equivalent is now provided by **reference** *INTEGER*, **reference** *CHARACTER* etc.

F.5 REMOVED CONSTRUCTS

The notion of *BIT* type has been removed. It enabled manipulation of bit sequences. The richer set of features in class *INTEGER* — *bit_and*, *bit_not* and so on, as well as the creation procedure *make* that sets the bit size to an arbitrary value — provides a more versatile replacement.

The notion of Strip expression has been removed. It was mainly useful in assertions and is advantageously replaced by a combination of tuple and agent mechanisms.

F.6 LEXICAL AND SYNTACTIC CHANGES

A small change to the method of language description, rather than the language itself: in the conventions for describing the syntax, a “zero or more” repetitition is now marked by an asterisk, as in {Type "**;**" ...}*, for symmetry with the convention for “one or more”, which uses a plus sign. Previously, the asterisk was omitted.

“Repetition productions”, page 32.

There are six new reserved words as already noted: **agent**, **create** (making a comeback from Eiffel 1 and 2), **pure**, *Precursor*, **reference**, *TUPLE*.

The words **creation** and **select** are no longer keywords (hence no longer reserved), but compilers will probably treate them as reserved for a while, the first as a synonym for **create**, the second to support previous repeated inheritance rules.

The following words are no longer reserved: *BOOLEAN*, *CHARACTER*, *INTEGER*, *REAL*, *DOUBLE*, *POINTER*. You should still not use them as class names, since they would conflict with classes that an Eiffel compiler will expect to find in the Kernel Library, and optimize. But you may now call a feature *integer* (although that's probably not a good idea).

An Index_clause is of the form

“INDEXING A CLASS”, 4.8, page 64.

something: *a*, *b*, *c*

where *something*: is the Index and one or more Index_terms follow the colon. Previously the Index part (including the colon) was optional. In practice developers included it almost all of the time. It is now required. This makes the grammar more regular, and facilitates parsing, especially as the semicolon is optional between an Index_clause and the next.

A syntax rule required underscores, if used in manifest integer and real numbers, to separate digits by groups of three. It has been replaced by a mere style recommendation.

“INTEGERS”, 29.13, page 699, and “REAL NUMBERS”, 29.14, page 700.

The syntax for creation instructions previously used exclamation mark characters **!**. For clarity, this has now been replaced by a keyword-based notation relying on the keyword **create**, permitted for creation expressions as well (see new constructs below). For consistency and to avoid any confusion, the keyword **create** is also used to introduce a Creators part listing the creation procedures of a class (previously the keyword there was **creation**).

The recommended separator between successive generic parameters, either formal as in a class declaration **class** *C* [*G*; *H*] … or actual as in a generic derivation *C* [*TYPE1*; *TYPE2*], is now the semicolon. The comma (the previous choice) is still supported.

Sections 12.2 and 12.3.

The Precursor construct, which may include an explicit type as in

```
Precursor {TYPE}          -- Or the version with arguments:
Precursor {TYPE} (arguments)
```

was first introduced in *Object-Oriented Software Construction, 2nd edition* (Prentice Hall, 1997), where this form of the construct is written with the type specification first: *{TYPE} Precursor* (…). An early printing even had double … braces, as in *{{TYPE}} Precursor* (…), showing once again that simple solutions sometimes come last. ISE Eiffel currently supports all three variants, but with the publication of this book the discarded ones should quickly disappear from practical use.

The syntax for New_export_item, in the New_exports clause that allows a class to change the export status of some inherited features, now supports an optional Header_comment to indicate the status of the corresponding features, such as -- Implementation. This is consistent with the corresponding convention for labeling feature clauses.

Examples in 7.11, page 130; syntax on page 136.

The new Feature Name consistency principle enables you, in any situation where you need a feature name, to use any of the three kinds: Feature_identifier, Operator_feature_name, Assignment_procedure_name. This improves the language's consistency, expressiveness and ease of use.

Page 87.

F.7 CHANGES IN VALIDITY CONSTRAINTS AND CONFORMANCE RULES

Some changes, most of them simplifications, have been brought to validity constraints (including conformance rules, treated in the same style as validity constraints in chapter 14). The changes are summarized in the following table.

Note that all constraint codes have been changed from *Vxyz* (*V* for validity) to *Cxyz* (*C* for constraint), to avoid any confusion to users while compilers are transitioning from Eiffel 3 to Eiffel 5. The table does not list constraints for which this is the only change.

See page 37 about this change.

A number of constraints have been **removed**, for one of three reasons:

- The constraint was found to be too restrictive, and its removal not to have any negative effect on software quality.
- The constraint was really a style rule, and users felt it should not be enforced by compilers.
- Other language changes made the constraint unnecessary.

A few constraints have been **added** to reflect the rules associated with the new constructs of Eiffel 5.

In addition, the table includes entries for some constraints having undergone changes affecting only their presentation:

- The order of clauses may have been changed for clearer exposition.
- Every constraint has a name; for consistency, some names have been changed (or added in the few cases of originally nameless constraints).

- Every constraint has a *Cxyz* code (previously *Vxyz*); in a few cases this has been changed for better mnemonic value and consistency. (The table, as noted, only lists a constraint if the *xyz* part has changed.)

Page numbers in *small italics* in the second column refer to the first edition of this book and determine the order of entries in the table.

Constraint name	**Old code,** *page*	**New code, page**	**Explanation**
Root Class rule	*VSRC* *36*	CSRC 46	Clause 2 added to preclude root class of a system from being deferred, necessary condition omitted in first edition. Removes limitation to one creation procedure. Previous clause 2 is now clause 2 of new constraint CSRP (next entry).
Root Procedure rule (previously covered by Root Class rule, see previous entry)	*VSRC* *36*	CSRP 47	New rule covering what was clause 2 of VSRC (previous entry). Previous phrasing, applying to *all* creation procedures of root class, was too restrictive. New rule applies to system's root procedure only. Clause 1 states that root procedure must be creation procedure of root class.
Cluster Class Name rule (previously: no name)	*VSCN* *51*	CSCC 51	Name added. Phrasing changed for consistency, but this does not affect rule's substance.
Class Header rule	*VCCH* *51*	CCCH 67	Loosened to permit the declaration of a class as deferred even if it has no deferred feature.
(No name)	*VCRN* *53*	*Removed*	Required ending comment of class, if present, to repeat class name. Changed into style rule. See *"ENDING COMMENT", 4.12, page 70*.
Target Conversion rule	(*NEW*)	CFTC 609	New rule defining validity of new notions: the optional conversion types of an infix feature.
Parent rule	*VHPR* *81*	CHPR 106	Clause 3 is new, to ensure CHUC (see next entry).
Universal Conformance rule	(*NEW*) *81*	CHUC 106	Theorem, follows from other validity rules. Was essentially satisfied before, but not stated.
Rename Clause rule	*VHRC* *81*	CHRC 110	Three new clauses: clause 3 requires Feature Name rule (CMFN, page 351) to apply (previously was only expressed as margin comment); clauses 4 and 5 cover case of renaming into infix or prefix feature.
Class *ANY* rule	*VHAY* *88*	CHCA 116	Code change for clarity.
(No name)	*VLCP* *101*	*Removed*	Required identifiers listed in a Clients part to be names of classes in the universe. See rationale for the removal in the paragraphs starting with "*There is **no validity constraint** on* Clients *parts*", page 135.
Entity Declaration rule	*VREG* *110*	CRED 145	Code change for clarity.
Old Expression rule	*VAOL* *124*	CAOX 162	Code change for clarity.

Constraint name	**Old code,** *page*		**New code, page**		**Explanation**
Precursor rule	(*NEW*)		CDPR	215	New constraint, covering new construct.
Definition of deferred and effective class		*161*		67	(Not validity constraint, but definition used by other constraints.) Moved to earlier chapter; updated to permit class to be deferred even without deferred features. See entry on *VCCH*/CCCH above.
Deferred class property		(*161*)		220	(Not separate constraint, but consequence of others.) Clarifies that a class can be deferred even without deferred features. See previous and next entries.
Effective class property		(*161*)		221	(Not separate constraint, but consequence of others.) Clarifies that a class can be deferred even without deferred features. See previous two entries.
Redeclaration rule	*VDRD*	*163*	CDRD	223	Last clause removed; prohibited redefining an external feature into an Internal one. This was an implementation constraint, no longer justified.
Join rule	*VDJR*	*165*	CDJR	225	Rephrased to take into account two cases missed by original: joining of one effective feature with one or more deferred ones; redefinition of all. Not language change but clarification of rule that was always there.
Join semantics rule (not validity constraint but semantic rule)		*166*		227	Beginning of rule updated to include cases mentioned in previous entry. Clause 6 added to cover case of effecting one or more deferred features.
Name Clash rule (previously: no name)	*VNCN*	*189*	CMNC	352	Name change for consistency. Slight rephrasing, but no change of substance. This is a redundant rule, following from *VMFN*/CMFN (Feature Name, unchanged).
Select Subclause rule	*VMSS*	*192*	*Removed*		Governed a clause, Select, that no longer exists thanks to simplification of repeated inheritance mechanism.
Constrained Genericity rule	*VTCG*	*203*	CTCG	262	Clause 3 amended to permit recursive constraints, as in **class** *C* [*G*, *H* –> *ARRAY* [*G*]]. New clause 4 addressing the new generic creation mechanism.
Expanded Type rule	*VTEC*	*209*	CTET	244	Code change for clarity. Clause 2 considerably loosened thanks to the new *default_create* convention.
Anchored Type rule	*VTAT*	*214*	CTAT	252	Rule rewritten with considerably loosened conditions: anchor chains now possible (*a* declared **like** *b* with *b* declared **like** *c*) as long as there is no cycle; anchoring now permitted on expanded and formal generic.
Direct conformance: class type reference			CNCR	290	New rule, covering conformance for new construct Class_type_reference.
Direct conformance: formal generic	*VNCF*	*224*	CNCF	294	Slightly reworded to address multiple constraints.

Constraint name	**Old code,** *page*		**New code, page**		**Explanation**
Direct conformance: expanded types	*VNCE*	*229*	CNCE	296	Previous clauses 2 and 3 removed as they are now covered by convertibility rather than conformance (in a more general form including new explicitly sized arithmetic types such as *INTEGER_16*). New clause 2 for reference/expanded symmetry (also a consequence of CNCR, see previous entry).
Direct conformance: Bit types	*VNCB*	*229*	*Removed*		No longer applicable since Bit types were removed.
Direct conformance: tuple types	(*NEW*)		CNCT	298	New rule, covering conformance for new kind of type.
Multi-branch rule	*VOMB*	*239*	COMB	363	Previous clause 6 removed; required Unique constants in Multi_branch to come all from same class. Removal means more work for compiler writers but more flexibility for programmers. Clause 1 now allows strings and type descriptors as inspect values. New clause 6 covers type descriptor case.
Unique Declaration semantics (previously: Unique declaration rule)		*266*		396	Name change for consistency with conventions used elsewhere (this is a semantic rule, not a consistency constraint). See next entry.
Unique Declaration rule (previously: no name)	*VQUI*	*266*	CQUD	396	Name and code change for consistency with conventions used elsewhere. See previous entry.
Entity rule	*VEEN*	*276*	CEEN	407	Clearer clause numbering; new clause 5 (imitated from clause 4) to cover new notion of inline agent.
Creation Clause rule	*VGCP*	*285*	CGCC	432	Code change for clarity. Previous clause 4 removed: made unnecessary by *default_create* convention; clause 2 of CTET takes care of the rest. New clause 4 added to preclude using once routines. Do not confuse with old *VGCC* (next entry).
Creation Instruction rule	*VGCC*	*286*	CGCI	436	Code change for clarity. Drastic simplification. Note that some of the old clauses reappear as "corollaries" of CGCI in the new CGCR, page 438. Do not confuse with new CGCC (previous entry).
(No name)	*VGCI*	*288*	*Removed*		System validity part removed. Do not confuse with CGCI (previous entry).
(Some clauses of *VGCC*)	*VGCC*	*288*	CGCR		New rule, corollary of CGCI (next-to-previous entry) and hence redundant, but providing extra error messages for compilers.
Assignment Attempt rule	*VJRV*	*332*	CJAA	491	Code change for clarity.
General Call rule (previously: Call rule)	*VUGV*	*367*	CUGC	536	Name change for consistency.

Constraint name	**Old code,** *page*	**New code, page**	**Explanation**
Single-Level Call rule (previously: no name)	*VUCS* *367*	CUSC 537	Code change; name added.
Argument rule	*VUAR* *367*	CUAR 538	Loosening of clause 4, which forbade constant attributes as Address (dollar sign) arguments. This is now permitted, as well as Parenthesized expressions. Semantics is to pass the address of a location containing value of constant or expression.
Manfest String rule (previously: no name)	*VWMS* *390*	CWMS 622	Previous clause 1 removed (belongs in the syntax). No more need for clause numbers. Symmetry between treatment of break characters after initial **%** and before final **%**.
Manifest Array rule	*VWMA* *393*	*Removed*	No longer necessary thanks to manifest tuples. Backward compatibility enforced through rule that manifest tuples conform to manifest arrays.
Identifier rule (previously: no name)	*VIRW* *418*	CIID 695	Code and name change.