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 Note clause (then Indexing), infix and prefix operators (now treated through **alias** clauses), the Obsolete clause, Unique values (removed in the present iteration), 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 *removes* a number of mechanisms (*BIT* types, Strip expressions), for which we identified better alternatives.

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.

The presentation of Eiffel 5 changes will successively consider: removed mechanisms; compatibility issues; new constructs; semantic changes to existing constructs; lexical and syntactic changes; changes to validity constraints and conformance rules.

F.2 REMOVED MECHANISMS

It has been a general principle of Eiffel evolution that in spite of its high expressive power the language should remain of manageable size, allowing Eiffel programmers to master *all* of Eiffel: there must be no dark holes in the language. In particular, if we find a better way of doing something, there is no reason to retain the previous constructs, as long as we make the transition easy for existing programs (see the compatibility notes in the next section). Along with its introduction of powerful new mechanisms, Eiffel 5 removes a few that are no longer needed.

The notion of infix and prefix features are now handled by a simpler and more general mechanism, using the existing keyword **alias**. The keywords **infix** and **prefix** are, as a consequence, no longer necessary. There is no loss of functionality — rather, a more general mechanism.

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.

Type *DOUBLE*, for "double-precision" reals, has been removed. The evolution of computer hardware and the needs of numerical computation lead to making every *REAL* 64-bit long. The new sized type *REAL_32* is available to declare shorter floating-point numbers.

The *global inheritance structure* has been simplified: *ANY* no longer has ancestors *GENERAL* and *PLATFORM*. *GENERAL* is gone, so *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.

F.3 BACKWARD 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**, **attached**, **attribute**, **create**, *Precursor*, **only**, **note**, *TUPLE*.

  The keywords **creation**, **indexing**, **infix**, **prefix** 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 *ANY*.

- 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 a Note clause (previously Indexing) the initial colon-terminated Note_name 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.

Any compatibility problem resulting from the removal of *GENERAL* and *PLATFORM* should be easy to correct.

F.4 NEW CONSTRUCTS

The *agent* mechanism (using tuples) is a major addition. *Chapter 27.*

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

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 550.*

Assigner procedures, allowing a procedure call *x.put (v, i)* to be written in assignment-like syntax as *x.item (i) := v* if *put* has been declared as an associated procedure for a query *item*, is new. ====

A related mechanism, *bracket syntax* for queries and commands, allowing the previous instruction also to be written *x [i] := v*, is new. ====

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_real*. 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. The notion of "*compatibility*", covering both conformance and convertibility, is a result of this addition; for assignment and argument passing, the rule is that the source type must be compatible with the target type, not just conforming as before. *Chapter 15.*

The Precursor construct is new, replacing techniques (still applicable in complex cases) relying on repeated inheritance. *10.24, page 293.*

The **only** postcondition clause, useful to avoid unwanted side effects especially in assertions and concurrent computation, is new. *Chapters 8 and 20.*

The use of a Note clause (previously Indexing) to annotate a feature, a control structure or the end of a class is new. Previously, Indexing clauses were applicable at the beginning of a class only.

The ability to declare an attribute explicitly, with the keyword **attribute**, is new. This allows attaching preconditions, postconditions and note clauses to attributes as well as routines. The previous syntax, just *x*: *A*, remains applicable as a common abbreviation.

Verbatim strings are new. .*29.8, page 784.*

The sized variants of basic types, such as *INTEGER_8* and *REAL_64* are new. ====

====

The ~ operator for object equality, associated with *is_equal*, is new.

38.5 SEMANTIC EXTENSIONS AND 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*]].

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 *ANY* and redefinable in any class.

Chapter 20.

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. A consequence is that it is no longer permitted to have an empty Creation_procedure_list in a Creation_clause; specifying **class** *A* **create feature** ... with nothing after **create** was a way to prohibit instantiating the class. It now suffices to make *A* deferred, even if all its features are effective.

Class Header rule, page 126; creation clause syntax, page 539.

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.

11.10, page 331.

The *Feature Identifier principle* is new in its full generality. The difference between operator and identifier features was and is intended for feature calls only; what is new is that every feature now has an associated identifier, with the infix, prefix or bracket alias providing only a simplification for calls. This convention doesn't just serve consistency, but also allows, for example, to define agents on features of any kind.

Page 153.

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", 23.14, page 633.

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", 17.4, page 474.

The arithmetic types have been developed and made more precise; this includes new types such as *INTEGER_8* noted in the previous section, but also the specification that *INTEGER* means 32-bit integer and *REAL* means 64-bit real, and also explains the removal of *DOUBLE*.

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", 21.6, page 572, and "COPYING AN OBJECT", 21.2, page 557.

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** {*NONE*} *C*, hereby providing a simpler solution to the issues of repeated inheritance and removing the need for Select.

"NON-CONFORMING INHERITANCE", 6.8, page 178.; "THE CASE OF REDECLARED FEATURES", 16.5, page 434

The possibility to declare a class — not just a routine — as *frozen* is new.

"CLASS HEADER", 4.9, page 124.

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 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.

Chapter 31.

F.5 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	FIX FIX FIX FIX!!!! Between arguments	Between target and argument
Redefinable	*equal* **alias** "}={" <—	*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.

---- FIX FIX FIX "~" 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*.

---- FIX FIX FIX 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

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 90.

There are eight new reserved words as already noted: **agent**, **attribute**, **create** (making a comeback from Eiffel 1 and 2), **note**, **only**, *Precursor*, **reference**, *TUPLE*. Among these, **create** is a replacement for **creation** and **note** for **indexing**.

The words **creation**, **note** 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).

A Note_entry is of the form

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

"ANNOTATING A CLASS", 4.8, page 122.

where *something*: is the Note_name and one or more Note_item follow the colon. Previously the Note_name 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 a Note_entry 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", 32.16, page 889, and "REAL NUMBERS", 32.17, page 892.

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 , page 200; syntax on page 205.

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.

Some of these changes involve a constraint that has 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 a 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*	VSRT	112	Clause 3 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 VSRP (next entry).
Root Procedure rule (previously covered by Root Class rule, see previous entry)	*VSRC* *36*	VSRP	112	New rule covering what was clause 2 of VSRC (previous entry). Previous phrasing, applying to *all* creation procedures of root class, was too restrictive. Clause 2 of new rule governs root procedure only. Clause 1 states that root procedure must be creation procedure of root class. Clause 3 is a new condition, prohibiting preconditions.
Cluster Class Name rule (previously: no name)	*VSCN* *51*	*Removed*		---- COMPLETE ----
Class Header rule	*VCCH* *51*	VCCH	126	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. Ending comment has been removed, even as a style rule.
Feature Declaration rule	*VFFD* *69*	VFFD	160	Replacement of clauses 5 and 6 by reference to Alias Validity (see next entry).
Alias validity	*VFFD* *69* (*Clauses 5 and 6*))	VFAV	162	Revision of part of VFFD accounting for new of **alias** clauses replacing **prefix** and **infix** and introducing bracket features.
Parent rule	*VHPR* *81*	VHPR	176	The rule now refers to the *Unfolded Inheritance Clause* of a class to account for implicit inheritance from *ANY*. Clause 2 is new, to take into account the new notion of frozen class. Clause 4 is new, to ensure VHUC (see next entry). Clause 5 should have been there all along but is new.
Universal Conformance rule	(*NEW*) *81*	VHUC	173	Theorem, follows from other validity rules. Was essentially satisfied before, but not stated.
Rename Clause rule	*VHRC* *81*	VHRC	182	Two new clauses: 3 requires Feature Name rule (VMFN, page 466) to apply (previously only expressed as margin comment); 4 covers renaming into feature with operator or bracket alias.

Constraint name	**Old code,** *page*	**New code**	**Page**	**Explanation**
Class *ANY* rule	*VHAY* *88*	VHCA	173	Code change for clarity.
Expanded Client rule	*VLEC* *94*	*Removed*		New semantics of expanded variables makes it possible to accommodate expanded client cycles.
(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 204.
Entity Declaration rule	*VREG* *110*	VRED	217	Code change for clarity.
Local Variable rule	*VRLE* *115*	VRLV	222	Code change for clarity (previous terminology was "Local Entity").
Feature Body rule (replacing Routine rule)	*VRRR* *113*	VFFB	144	New rule is generalization of old one: covers all features, not just routines. It follows from the introduction of the **attribute** keyword, making some clauses (in particular Precondition and Postcondition) applicable to all features.
Old Expression rule	*VAOL* *124*	VAOX	235	Code change for clarity.
Old Expression rule	(*NEW*)	VAON	240	Validity rule for new **only** construct.
Precursor rule	(*NEW*)	VDPR	298	New rule , covering new construct.
Definition of deferred and effective class	*161*		127	(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 above.
Deferred class property	(*161*)		304	(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*)		305	(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*	VDRD	307	Last clause removed; prohibited redefining an external feature into an Internal one. This was an implementation constraint, no longer justified.
Join rule	*VDJR* *165*	VDJR	309	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*		312	Beginning of rule updated to include cases mentioned in previous entry. Clause 6 added to cover case of effecting one or more deferred features.

Constraint name	Old code, *page*	New code	Page	Explanation
Name Clash rule (previously: no name)	*VNCN* *189*	VMNC	467	Name change for consistency. Slight rephrasing, but no change of substance. This is a redundant rule, following from *VMFN*/VMFN (Feature Name, unchanged).
Select Subclause rule	*VMSS* *192*	*Removed*		Governed a clause, Select, that no longer exists thanks to simplification of repeated inheritance mechanism.
Unconstrained Genericity rule	*VTUG* *201*	*Removed*		Now merged with VTGD of which it was a special case (repeated in its clause 1).
Generic Constraint rule	(*NEW*)	VTGC	349	New rule taking into account generic creation and multiple generic constraints.
Genericity Derivation rule (previously: Constrained Genericity rule)	*VTCG* *203*	VTGD	351	Clause 2 amended to permit recursive constraints, as in **class** *C* [*G*, *H* –> *ARRAY* [*G*]].
Expanded Type rule	*VTEC* *209*	VCCH	126	Rule no longer needed as type rule thanks to removal of **expanded** *T* types (all expanded types are now based on an expanded class) and removal of requirement of *default_create* for expanded types.
Anchored Type rule	*VTAT* *214*	VTAT	337	Considerably loosened conditions: anchor chains now possible (*a* declared **like** *b* with *b* declared **like** *c*) if there's no cycle; anchoring now permitted on expanded and formal generic. No more anchoring on arguments. Properties of anchored type now completely defined by those of its unfolded form.
General conformance	*VNCC* *219*	VNCC	380	Clause 3 integrates attached type requirements; new clause 6 handles anchored types and allows removal of *VNCG* (see below).
Direct conformance: class types	*VNCN* *221*	VNCN	382	Simplified thanks to the notion of generic substitution; also subsumes *VNCG* (next entry).
Direct conformance: generic substitution	*VNCG* *222*	*Removed*		Covered by new formulation of VNCN (see previous entry).
Direct conformance: formal generic	*VNCF* *224*	VNCF	385	Simplified thanks to a more general notion of constraint. Also, addresses multiple constraints.
Direct conformance: anchored types	*VNCH* *225*	*Removed*		Anchored types are now treated more simply like "macros". See clause of
Direct conformance: expanded types	*VNCE* *229*	VNCE	388	---- FIX --- 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*).
Direct conformance: Bit types	*VNCB* *229*	*Removed*		No longer applicable since Bit types were removed.

Constraint name	**Old code,** *page*	**New code**	**Page**	**Explanation**
Direct conformance: tuple types	(*NEW*)	VNCT	389	New rule, covering conformance for new kind of type.
Conversion Procedure rule	(*NEW*)	VYCP	403	Convertibility is new.
Conversion Query rule	(*NEW*)	VYCQ	405	Convertibility is new.
Expression convertibility	(*NEW*)	VYEC	415	Convertibility is new.
Precondition-free	(*NEW*)	VYPF	417	New concept closely connected with convertibility.
Multi-Branch rule	*VOMB* *239*	VOMB	480	Removed all constraints relating to Unique values, no longer present in the language.
Unique declaration rule	*266*	*Removed*		Removed all constraints relating to Unique values, no longer present in the language.
Unique Declaration rule (previously: no name)	*VQUI* *266*	*Removed*		Removed all constraints relating to Unique values, no longer present in the language.
Entity rule	*VEEN* *276*	VEEN	505	Clearer clause numbering; new clause 7 (imitated from clause 6) to cover new notion of inline agent.
Variable rule	(*NEW*)	VEVA	506	New rule made necessary by inline agents.
Creation Precondition rule	(*NEW*)	VGCP	539	New rule restricting what's permissible in the precondition of a creation procedure.
Creation Clause rule	*VGCP* *285*	VGCC	540	Code change for clarity. Previous clause 4 removed: made unnecessary by *default_create* convention; VCCH takes care of the rest. New clause 4 added to preclude using once routines. New clause 5 to rule out unsound precondition clauses. Do not confuse with new *VGCP* (previous entry) or old *VGCC* (next entry).
Creation Instruction rule	*VGCC* *286*	VGCI	545	Code change for clarity. Drastic simplification. Note that some of the old clauses reappear as "corollaries" of VGCI in the new VGCP, page 547. New clause 4 takes into account generic creation. Do not confuse with new VGCC (previous entry).
(No name)	*VGCI* *288*	*Removed*		System validity part removed. Do not confuse with clause now called VGCI (previous entry).
Creation Instruction Properties	(*Parts of VGCC*) *288*	VGCP	547	New rule, corollary of VGCI (next-to-previous entry) and hence redundant, but providing extra error messages for compilers.
Creation Expression rule	(*NEW*)	VGCE	553	Creation expressions are new.
Creation Expression properties	(*NEW*)	VGCX	554	Same relation to VGCE as VGCP to VGCI (see previous entries).

Constraint name	Old code, *page*	New code	Page	Explanation
Assigner Call rule	(*NEW*)	VBAC	602	Assigner calls are new.
Assignment Attempt rule	*VJRV* *332*	*Removed*		No more assignment attempt (replaced by Object_test)
Non-Object Call rule	(*NEW*)	VUNO	623	Non-object calls are new.
Call Use rule (previously: no name)	*VKCN* *368*	VUCN	615	Code change for consistency.
Export rule	*VUEX* *368*	VUEX	624	Simplification (the former case 2 wasn't necessary) and addition of Non_object_call case.
Argument rule	*VUAR* *369*	VUAR	626	Rule simplified thanks to the addition of VUDA (see below) for the more complex case. Clause 3 (redundant) removed. Clause 4 moved to constraint on Address expression.
Class-Level Call rule	(*NEW*)	VUCC	628	Separating class validity from more complex rules.
Object Test rule	(*NEW*)	VUOT	651	New rule, covering new construct.
Descendant Argument rule	(*VUAR, p. 367)* *367*	VUDA	659	Rule split away from VUAR to separate more advanced cases from simple ones.
Single-Level Call rule (previously: no name)	*VUCS* *367*	VUSC	660	Code change; name added.
General Call rule (previously: Call rule)	*VUGV* *367*	VUGC	673	Name change for consistency.
(No name)	*VWEQ*	*Removed*		No more conformance constraint on equality.
Call Agent rule	(*NEW*)	VPCA	746	Agents are new.
Inline Agent rule	(*NEW*)	VPIA	747	Inline agents are new.
Inline Agent requirements	(*NEW*)	VPIR	748	Inline agents are new.
Bracket Expression rule	(*NEW*)	VWBE	772	Bracket expressions are new..
Manifest Type rule	(*NEW*)	VWMQ	781	Manifest types for expressions are new..
(No name)	*VWMS* *390*	*Removed*		Now handled through syntax and definition of Line_wrapping_part.
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*	VIID	881	Code and name change.