AIRE/CE:
Advanced Intermediate Representation with Extensibility / Common Environment

Internal Intermediate Representation (IIR) Specification
Version 4.6 (Trial Implementation Draft of 2/3/99)
Including Digital VHDL & VHDL-AMS support

John Willis, Technical Editor / Architect (FTL Systems, Inc.)
Based on the review comments by:
Philip A. Wilsey (University of Cincinnati), Dale Martin (University of Cincinnati), Malolan Chetlur (University of Cincinnati), Rob Newshutz (FTL Systems, Inc.), Tim McBrayer (University of Cincinnati), Hal Carter (University of Cincinnati), Vasudevan Shanmugasundaram (University of Cincinnati), Steve Rogerman (FTL Systems, Inc.), Captain Gregory Peterson (USAF), Serafin Olcoz (SIDSA), Chuck Swart (Analogy), Steve Gregor (FTL Systems, Inc.), and Patrick Gallagher (FTL Systems, Inc.)

This technology is funded in part by U.S. Defense Advanced Research Projects Agency (DARPA), U.S. Air Force's Wright Laboratory and FTL Systems, Inc.

HTML copies of this document are available on-line from:
http://www.vi.org/vi/aire/index.html
http://www.ftlsystems.com/index.html/
http://www.eia.org/...
http://www.ece.uc.edu/~paw/aire/
...

ISBN *TBD*

Acknowledgments...

For more than ten years Dr. John Hines, Technical Director of System Concepts and Simulation Division under the Avionics Directorate at the US Air Force's Wright Laboratory, has advocated a common intermediate representation and associated component tools. His vision allows individual groups to concentrate on what they did best, rather than forcing each group to develop an entire VHDL development environment.

Captain Greg Peterson, at the US Air Force's Wright Laboratory, is responsible for key technical review of this specification and coordination of the standardization process. Thanks to Lt. Col. Michael Mills for his reviews.

This specification is developed in part through funding by the USAF's Wright Laboratory under contract F33615-95-C-1638 (with MTL and The University of Cincinnati). The authors appreciate Wright Laboratory's research support enabling AIRE and the personal efforts of Dr. John Hines and Dr. Greg Peterson.

Program managers Dr. Robert Parker, Randy Harr and Sonny Maynard, at The U.S. Government's Defense Advanced Research Projects Agency (DARPA), make AIRE possible through their encouragement, logistical support and research funding.

This specification was developed in part through funding by DARPA under contract DABT63-96-C-004 (with FTL Systems, Inc.) and by DARPA under contract J-FBI-93-116 (with MTL and The University of Cincinnati). The authors appreciate DARPA's research support enabling AIRE/CE and the personal efforts of Dr. Robert Parker and Randy Harr.

Documentation is provided by FTL Systems, as driven by a design-by-consensus process open to all parties actively using the specification. Substantial private funding was used in the development, editing and distribution of this specification.

VHDL International is providing computer resources required to maintain mail reflectors related to AIRE/CE as well as the single point of reference required for ISO 9000 compliance. Thanks to Randy Harr for providing this critical supporting resource.

AIRE/CE's technology is based on Auriga (developed at Carnegie-Mellon University and FTL Systems, Inc.), SAVANT (developed at the University of Cincinnati) and ICE (developed at TGI). Techniques for application-specific extensions to the class hierarchy come directly from the SAVANT research. Techniques for portability and database compression largely came from Auriga, support for Eiffel came from SIDSA's ICE.

CHAPTER 1 Introduction

Ten years after VHDL and Verilog were introduced to the hardware and system design community, design tools are still hand-crafted around proprietary software interfaces. Component tools from different development groups must generally either communicate via source code or not at all. Even large users are unable to readily mix and match analyzer, elaborator, optimizer, code generator, simulation, graphics and synthesis tool components. This situation is not in the best interests of the HDL design community.

The **A**dvanced **I**ntermediate **R**epresentation with **E**xtensibility / **C**ommon **E**nvironment (**AIRE/CE**) specifications address the HDL user's need for efficient mechanisms for sharing of design information between tool components. The **AIRE/CE** specifications includes coordinated internal intermediate representation (**IIR**) and file intermediate representation (**FIR**) specifications. This document specifies the IIR; a related document the FIR.

Existing tools, by their very nature, are already tied to proprietary representations and generally have little ongoing development resource available with which to change the underlying design representation. For practical reasons, such tools are unlikely candidates for use of a new intermediate representation, such as AIRE.

AIRE targets tools beginning early in the development cycle or at a major system redesign. AIRE's designers have concentrated on providing powerful new capabilities intended to facilitate system design well into the next century rather than operating within the tight constraints of backward compatibility. AIRE delivers on key user requirements including performance, capacity, portability, extensibility, security, availability and reusability; keys to advanced system design.

This chapter introduces AIRE's internal intermediate representation through a discussion of this document's purpose, earlier related work, AIRE's design approach, an overview of the specification's remaining chapters and an introduction to the specification change process. Hypertext renderings of this document provide linkages, beginning with this chapter, into the remainder of the document.

As an extensible representation, your version of this document may include non-core extensions. For example, extensions may support VHDL language extensions, other languages or implementation-specific information. Additions to the core AIRE representation are the result of a careful coordination process open to all those using this intermediate representations. All such extensions are denoted by underlined text, as in <u>extension</u>.

1.1 Purpose

This document intends to precisely specify characteristics of an internal intermediate representation (IIR). This representation defines a common representation for system design information while it is present in a random-access address space; typically a computer's virtual memory.

Through compliance with this specification, a development group should be able to design and develop inter-operable implementations of the intermediate representation (foundation) or component tools which work with information present in the internal intermediate representation (applications).

The IIR specification intentionally avoids over-constraining implementation decisions or artificially limiting implementation options. The specification defines the minimal set of interface constraints needed for effective inter-operability. Implementations are free to add both foundation-specific and application-specific information to the representation. By design, such extensions should retain a high degree of inter-operability with existing tools.

IIR performance characteristics and limitations associated with a specific IIR implementation are not part of this specification. For example, an implementation undoubtedly will make performance trade-offs between capacity and various kinds of object access and will constrain the total number of IIR objects which can be represented in a given memory. Such implementation-specific information should be found in an implementation-specific document, not in this specification.

The same AIRE/CE IIR representation is useful following source code analysis, integration of separately analyzed units, elaboration, in-line expansion, machine-independent optimization, backend code generation/synthesis and execution (simulation). Integration of component tools from different development groups should finally be realizable with modest effort using this intermediate representation. As a result both the research and design community benefit from stronger and more useful tools.

AIRE/CE's purpose is to meet the evolving needs of advanced tool developers. Therefore, the specification is still evolving. Some changes to the core specification are still being made as a result of experience with early implementations. Implementation-specific extensions are being added to the core specification. For the latest information on AIRE, please join the AIRE mail reflector by sending a request to:

aire-request@vhdl.org

and tracking on-line copies of the AIRE specifications via the following WWW sites:

http://www.vhdl.org/vi/aire/index.html
http://www.ftlsystems.com/aire/index.html
http://www.eia.org/...
http://www.ece.uc.edu/~paw/aire/
...

1.2 Related Efforts

The richness of Ada and its HDL derivative, VHDL, have lead to substantial earlier research and development efforts resulting in intermediate forms. AIRE's developers are fortunate enough to be able to learn from these early efforts without being constrained to achieve compatibility with any.

Intermediate representations have been moderately successful in facilitating Ada compiler implementations. DIANA[] serves as the basis for Ada compilers from the Ada Group Ltd.[]., Bell Labs[], Burroughs[], The University of California[], Berkeley[], Intermetrics[], The University of Karlsruhe[], Rational[], Rolm[], SoftTech Microsystems[], Stanford[], and Verdix[].

IVAN[] was initially conceived, in 1985, as the corresponding interface description language IR for VHDL. Unfortunately IVAN was not nearly as successful as DIANA. It never saw wide-spread use in the development of VHDL compilers beyond the initial Intermetrics VHDL implementation Relatively few implementations of IVAN are believed to remain in use and no new development efforts are believed underway using IVAN

In the late 1980s, CAD Language Systems (CLSI) developed an IR and associated VHDL analyzer. This analyzer achieved substantial commercial success and is still in use by some VHDL compilers. CLSI's intermediate representation is proprietary and limited to representation of IEEE Std. 1076-87. Experience with CLSI's intermediate identified several intrinsic design decisions which were later recognized to limit an implementation's performance.

For several years, ending in 1991, an IEEE Design Automation Standards Committee (DASC) Working Group attempted to develop VIFASG []. This effort is generally believed to be closely patterned after CLSI intermediate. Despite substantial effort by the working group, a standard failed to materialize and the IEEE has withdrawn the associated PAR. The group's inability to reach a standard is generally attributed to competing commercial concerns and sub-optimal technical characteristics.

The unsuccessful VIFASG effort was picked up by Leda SA, updated to represent IEEE Std. 1076-93, and is currently the basis for an analyzer product by Leda []. While this interface has seen some commercial success, it has less than optimal compaction, leads to relatively slow tools, does not benefit from IDL advances over the last five to ten years and does not address the complete design life-cycle of VHDL and Verilog designs. Leda's interface is supported by an expensive, single-source implementation. Until at least recently, Leda's interface has also been proprietary.

By the middle of 1995, HDL "power" users and advanced tool developers recognized that a fundamentally new intermediate representation was needed. Today, AIRE offers a widely-available, high-capacity, high-performance intermediate which takes maximal advantage of contemporary, object-oriented software techniques. Multiple IIR implementations are available, including a free non-commercial implementation and at least one low-cost commercial implementation. Over a dozen advanced applications using AIRE's IIR foundation are in development.

1.3 AIRE Approach

AIRE uses a collection of object (record or structure) instances linked by pointers to represent design information. These objects represent analyzed, elaborated, and executable instances of specific design information. In very general terms, the collection of objects represents a very generalized abstract syntax tree (AST), while methods associated with the classes (and thus objects) represent an integrated application programming interface (API).

The IIR class hierarchy is shown abstractly in Figure 1 on page 21. A base class, called **IIR**, is located at the top of the class hierarchy. Derived classes, descended from the base IIR class, are instantiated in order to form a specific design representation. Objects of a specific class are differentiable by an enumerated value denoting the object's IIR_Kind.

Methods predefined in the IIR class hierarchy may either be virtual or non-virtual (*this is an implementation-dependent property of the IIR foundation*). Virtualization of methods allows overloading by extension classes and easier pairing of objects with methods. Non-virtualized implementations permit faster and more compact implementations at the expense of additional software development effort.

IIR-derived class may be instantiated by other IIR classes (such as declaration lists within a block declaration), by friends of an implementation-specific foundation (such as a source code analyzer intrinsic to the foundation), or by external application code (such as the optimizer and code generator shown in Figure 1 on page 21). The public methods and public data elements are sufficient to construct, get and set all language functionality defined by the associated system design language.

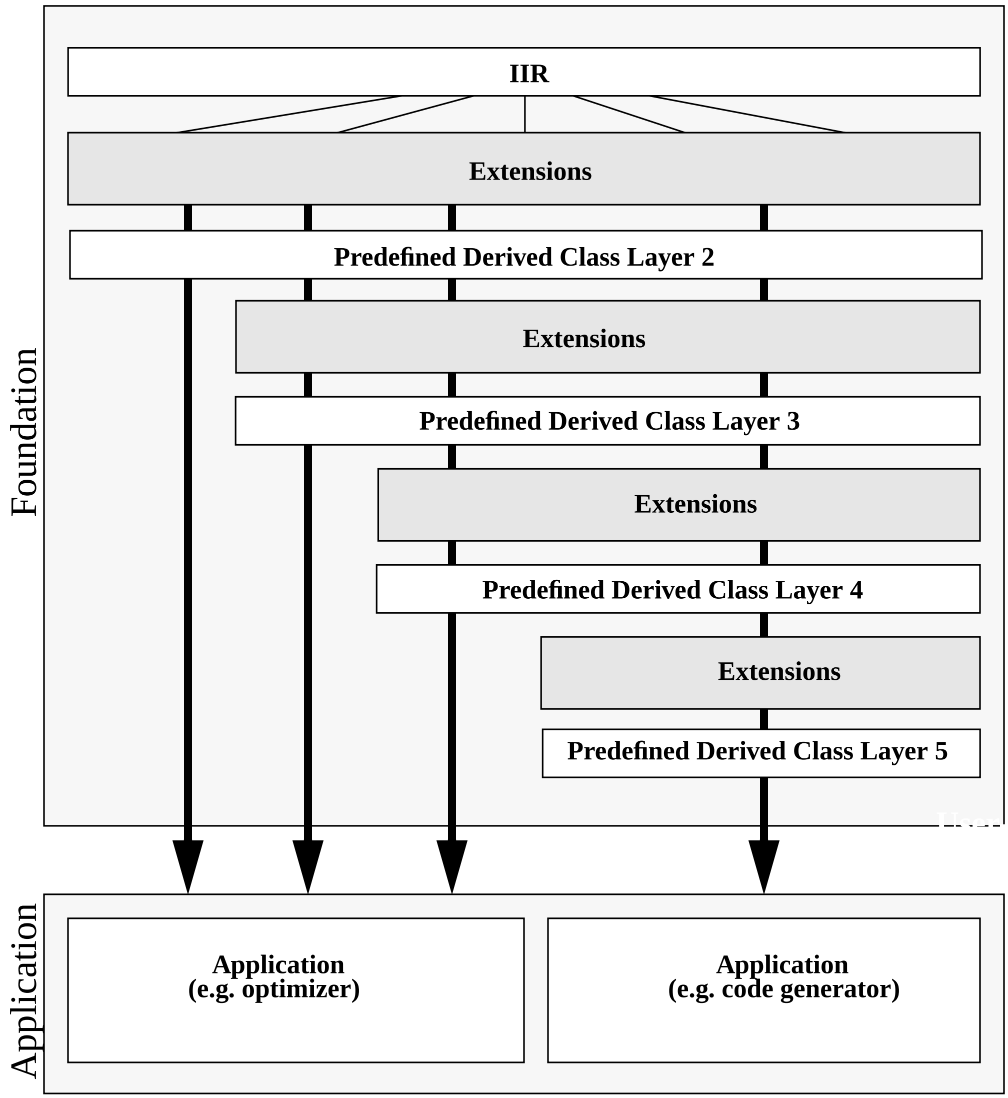
Extension classes, shown in darker shading in Figure 1 on page 21, may be interposed within the IIR inheritance hierarchy in order to add application-specific methods or (in instantiable classes) additional, application-specific data elements. Each extension layer begins derivation with a class named by pre-pending **IIRBase_**. The extension layer(s), if present, result in a final, predefined layer named by pre-pending **IIR_** to the class name. The predefined properties of this layer are explicitly specified in this document; the predefined properties of the base layer for each class may be readily (even automatically) derived. The names of any intervening extension classes should assume the form **IIR** <Extension Designator> '_' <Specific class name>. For example, a synthesis application might interpose an extension class **IIRSyn_ProcessStatement** between **IIRBase_ProcessStatement** and **IIR_ProcessStatement**.

Since a complete tool may be composed of more than one application, the darker shaded extension layers actual represent zero or more layers within the class hierarchy. For example, an elaborator application may add some extensions to the block and process declarations while a code generator adds its own extensions. In order to clearly delineate identifiers other than those predefined, constructructors, destructors or operators, such identifiers are prefaced by a leading underscore ('_'). Name conflicts are inevitable when linking two applications which both use the same extension designator, such as **Syn** in the example above.

The extension class approach was derived from the Savant project at the University of Cincinnati []. Savant defines extensions for publishing and transmuting classes. For example, a concurrent assertion statement may be transmuted into an equivalent process statement.

Whereas the specification is generally based on the C++ language and single-inheritance C++ classes, implementations using C, Ada or Pascal are very feasible. Chapter 18 describes how to extend AIRE's approach C++ so that AIRE-compliant tools may be developed in other programming languages such as C, Ada or Pascal. However since AIRE takes advantage of modern, object-oriented programming techniques which may not be present in older programming languages, AIRE can be a somewhat less convenient when other languages are used.

FIGURE 1. Structure of Internal Intermediate Representation (IIR) and applications



1.4 Document Structure

The remainder of this specification describes requirements for and details of the internal intermediate representation. The material may loosely be divided into requirements, basic data types, class hierarchy, hierarchy details (many chapters) and usage from languages other than C++.

Chapter 2 details design requirements which shape AIRE/CE's IIR. All of the subsequent design decisions are based on the explicit design objectives set forth in this chapter. The interested reader will see a close linkage between these design requirements and the decisions which follow.

A few basic data types, described in Chapter 3, define the data types used by methods accessing information within specific classes. These basic data types include integers, characters, and floating point values with specific ranges.

Chapter 4 presents the complete predefined IIR class hierarchy. Tables in this chapter allow the reader to trace inheritance relationships from the IIR base class through to derived classes from which instances may be created. This chapter also describes general properties of all IIR classes.

Details of the IIR base class and IIRKind enumerated type are found in "IIR Base Class" on page 61. As the base class, methods associated with the IIR class must be supported by all derived IIR classes. The IIRKind enumerated type allows distinguishes the class associated with dynamically allocated IIR objects.

Chapter 6 through Chapter 15 describe the IIRKind, predefined public methods, predefined public data and other characteristics associated with the various classes descended from the IIR base class. Each chapter begins with a table illustrating the class derivation hierarchy described by the following chapter. Then distinct sections describe each intermediate and terminal derived class.

Although AIRE/CE's IIR is specified in terms of C++ classes, compliant foundations and applications may be developed using other programming languages such as C, Ada and Pascal. Chapter 18 describes how to interpret the IIR specification in terms of other languages.

Finally, Chapter 19 records changes made to this document from one version to the next. This log begins with the trial implementation draft (Version 2.3).

On-line copies of this specification provide hypertext linkages, generally highlighted by a viewer or browser. These linkages allow the reader to rapidly navigate from use of a particular identifier back to its definition. In a like fashion the reader may directly step from a table of contents or index entry to the corresponding point in the document's body. Since these linkages greatly facilitate usage of the specification, access to an on-line version of this specification is strongly recommended.

1.5 Change Review Process

Changes and extensions to AIRE/CE are accomplished by a consensus-process open to all those implementing substantial tools compliant to AIRE/CE. Changes to a complex, shared document involving multiple implementations with distinct (and often conflicting) requirements are never easy. It is always easier for someone to "do their own thing" at the price of inter-operability. A commitment, by all parties involved in AIRE, to real inter-operability and fully open communication is essential.

In order to accelerate the update process, accommodate world-wide involvement, and reduce costs, the AIRE change review process is based on effective use of internet mail reflectors and WWW sites, not on face-to-face meetings.

The applicable language reference manual and language disambiguation authority, such as ISAC, represent the reference point defining language semantics.

The process for making a change to this document involves:

1. Change request sent to *aire@vhdl.org* (a more specific mail reflector may be needed for the review board). This request need not follow any specific form, but should state as specifically as possible what change is needed and ideally why.
2. Comment period sufficiently long for all interested parties to identify critical pros and cons of the proposed change. Conversely, keeping the comment period open too long delays the update process. The minimum period for comment should be fixed, perhaps at 2 weeks. A moderator is essential to keep this process on track and establish a closure point.
3. Precise statement of the document change proposed, circulated to the CRB reflector.
4. Consensus-based decision which respects all implementations involved, if needed at the cost of increased specification and tool implementation complexity. This is the price of inter-operability.
5. Specification editor will update the applicable text, maintaining change bars in the margin for at least 6 months. FTL Systems, Inc. is committed to provide specification documentation and editing services for AIRE through 1999. All editorial changes to the document are the result of following the above steps.
6. Update of FTP and WWW sites maintaining the specification.

The IIR specification will be placed under change review control following completion of the first two trial implementations. The change review process is intended to satisfy ISO 9000 compliance requirements. Changes will be made to the above process as needed to comply with ISO 9000.

CHAPTER 2 *Design Requirements*

This chapter describes the significant design requirements shaping the IIR specification. Specific design requirements related to language scope, representational domain, portability, extensibility, efficiency, integration, security, and availability.

2.1 Language Scope

IIR was initially designed to represent VHDL (IEEE Std. 1076-87 [1], IEEE Std. 1076-93 [2], and the upcoming IEEE Std. 1076-98). Extensions under development by FTL Systems support IEEE PAR 1076.1 (Analog VHDL) [3], IEEE Std. 1364 (Verilog) [4], and ANSI/ISO C++ [5].

Class and method names are chosen so as to correspond as closely as feasible with the corresponding language standard without violating the original standard's copyright. As one ramification of this design decision, the class and method names tend to be longer than would otherwise be chosen.

VHDL serves as the semantic basis for all extensions currently under development. This breadth facilitates practical design environments using an integrated mixture of VHDL, Verilog and C++ (co-design). By sharing a common IIR, designs represented in one or more of the languages may be more tightly, efficiently and rapidly compiled together.

2.2 Representation Domain

AIRE's designers believe that an IIR must represent HDL-derived design information through its entire lifecycle. This life cycle typically begins with source code analysis. Conventional intermediate representations often end following analysis. IIR is intended to continue on through elaboration, optimization, backend generation (embedded code, simulation or synthesis), execution (runtime or simulation) and graphical display.

2.3 Portability

A key IIR objective is to sufficiently specify the IIR interface that either component tools using the interface (applications) or implementations of the IIR may be coded and inter-operate without further co-ordination (beyond this specification). Conversely, the IIR specification should avoid placing non-essential requirements on an IIR implementation or application. Over-specification tends to reduce the diversity of performance trade-offs and compatible tools.

The IIR is also intended to be portable to a variety of processor and operating system architectures. Processor instruction set architectures may vary in the precision and layout of intrinsic data types as well as address space organizations. Portability requires that the IIR specification can be effectively matched to any of today's mainstream processor architectures and anticipate characteristics of those likely to be in common use within the next 5 to 10 years.

Although C++ is intended as the primary tool development language, it is imperative that IIR be accessible from a variety of mainstream programming languages including C, Ada and Pascal. Many useful tools have and will be developed using languages other than C++.

Distribution of system design tools takes many forms. Many university-developed tools are distributed in source form. Commercially developed tools tend to be distributed in executable (binary). A critical design requirement for this work was the need to support both foundation (IIR implementation) as well as application tools distributed in either source or executable forms.

2.4 Extensibility

Functionality required by useful tools typically increases over time. Especially in a research environment, a successful IIR must provide a means by which new information or functionality can be associated with an existing description or entirely new kinds of informational structures can be represented on the spur-of-the-moment. In order to maximize performance and capacity, the ability to allocate space for extension information within the same storage unit as core information is important.

In order for such an extensible IIR to also remain portable, the format needs to have a well-defined, stable, and common core functionality; it must also have extensibility mechanisms allowing a core-compliant tool to ignore non-core information with minimal impact.

In order to maintain and evolve a core which meets the needs of a broad user community, it is important that there be an open but controlled process for changes to the core IIR. Control by an independent AIRE Change Review Board (CRB), comprised of those actively using the specification in substantial tool development or with substantial motivation for AIRE's success, is critical to the AIRE's long term success.

2.5 Efficiency

A production-quality IIR must be much more efficient (in both space and time) to read and write than reading or writing the equivalent HDL source code. Both space and time efficiency dictate a binary (rather than strictly textual) IIR. The binary representation must be compact, storing the required information with minimal redundancy.

Objects within the IIR must be rapidly mapped into and out of the corresponding file intermediate representation (FIR) []. Whereas a direct map between IIR and FIR, such as a memory-mapped file, provides high performance, such a technique does not satisfy our portability or integration requirements.

Space and often time requirements dictate use of canonical elements wherever possible. For example, there is only one representation of the identifier “foo” in a given memory image. All uses of “foo” refer back to the common, canonical representation of the identifier.

2.6 Integration

A practical IIR must facilitate integration of separately compiled units and extraction of design fragments for re-integration with other designs. The units may include predefined language components, library components, or separate design sub-units integrated by a design team.

In order to accommodate rapidly developing standard and locally standard packages, the IIR must avoid “hard-wiring” standard definitions, yet facilitate merging of separately “pre-compiled” designs referring to a common set of standard or utility libraries.

2.7 Security

Designs often embody proprietary information and intellectual property. Typically some subset of this information must be exported in order to make the design useful. This exported design information must be usable for exactly what the information supplier intended; no more and no less.

Today, there are no openly available mechanisms by which intellectual property owners may implement discretionary export of their design information. The Open Modeling Forum (OMF) is working to specify a means of distributing fully compiled component models. The OMF does not provide for representation of partially compiled design information; tools are unable to optimize across the caller/callee boundary of a model and are unable to retarget a model; the model is pre-compiled for simulation on a specific target architecture.

Without some means of providing discretionary security within the IIR, intellectual property owners are unwilling to export their design information. Since compiled forms of intellectual property have a well-defined standing in the legal community, it is important that the IIR be considered as a compiled form of the design and not a wrapper around the design’s source code.

IIR must be capable of representing as little or as much of the designer's original source code as the designer intends without structural or semantic changes to the underlying IIR definition. A highly obscure representation will remove all meaning from identifiers, elaborate, expand and obfuscate the design. Component tools operating on the IIR must still function on the obscured form of the design. Conversely, some designers require that the IIR contain a loss-less representation of the original design, including comments and source layout. For example, such a complete representation facilitates short-term symbolic debug and long-term documentation of design intent. AIRE's designers intend that the IIR embrace this full usage range.

2.8 Availability

AIRE's designers believe that the core IIR specification must be readily available at little (or no) cost to anyone implementing foundation or application tools compliant to the specification. Furthermore, no royalty or other cost must be imposed on tools solely for using this specification.

In order to facilitate AIRE's use, a snap-shot of this specification is available in HTML on the world-wide web and Postscript via anonymous FTP. Such network-based publishing increases availability of the specification while discouraging local modification of the core document. Users are welcome and encouraged to provide hypertext links into their own, clearly distinguished extensions. AIRE's portability disappears if evolution of the core specification is not globally synchronized by a change-control board.

Even more important than formal standardization is the availability of multiple, high-quality, affordable implementations of IIR foundations and applications are critical to AIRE's success. Already, two implementations of the foundation are underway and nearly a dozen applications. At least one university implementation is free for non-commercial use and available for license with commercial support. A second, commercial implementation is geared toward high capacity, parallel tool implementation. Concurrent development of multiple implementations helps to insure that AIRE is sufficiently specified without becoming design documentation for any one implementation.

2.9 Bibliography

[1] IEEE Standard VHDL Language Reference Manual, IEEE Std. 1076-1987.

[2] IEEE Standard VHDL Language Reference Manual, IEEE Std. 1076-93, ISBN 1-55937-376-8.

[3] IEEE Standard VHDL Language Reference Manual (Integrated with VHDL-AMS changes), IEEE Std. 1076.1 (Proposed).

[4] IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language, IEEE Std. 1364-1995, ISBN 1-55937-737-5.

CHAPTER 3

Basic Data Types

A small set of basic data types, specified in this chapter, lend precision and portability to constructing, getting and setting IIR values. These basic data types include:

- IR_Boolean,
- IR_Char,
- 32-bit integers,
- 64-bit integers,
- single precision floating point,
- double precision floating point,
- IR_Text,
- IR_Kind,
- IR_SignalKind,
- IR_Mode,
- IR_Pure,
- IR_DelayMechanism,
- IR_SourceLanguage,
- IIR pointers.

Compliant implementations of the IIR should insure that the visible type definitions are compatible. The definitions may be via compatible language predefined types, included system header files, primitive type definitions within the IIR header file(s) or class definitions within the IIR header file(s).

In some ways lists also act as a basic data type. Like basic data types, lists are directly instantiated within IIR classes. Also like basic data types, lists cannot have additional data elements added. However unlike basic data types, lists are generally not directly supported by machine data types within an instruction set architecture. For further discussion on IIR lists, please see “IIR_List Derived Classes” on page 135.

3.1 IR_Boolean

The boolean type must use the identifier ‘**IR_Boolean**’ with two enumerated values: ‘**FALSE**’ and ‘**TRUE**’. The type is either an enumerated type (preferred) or an integer with range including 0 and 1. Values may either come from an enumerated type (preferred) or the constants 0 (FALSE) and 1 (TRUE).

3.2 IR_Char

A character type must be provided called **‘IR_Char** with a domain of values which at least includes those defined by ISO 8859-1 : 1987. Informally, ISO 8859-1 : 1987 defines a character set with 256 distinct encodings within 8 bits. Implementors should pay careful attention that ISO 8859-1 defines a significant character with the value 0; 0 is not available to uniquely denote the end of a string.

3.3 32-bit Integer

A 32 bit integer type must be provided called ‘**IR_Int32**’ with a domain of discrete values including the range -2,147,483, 647 to +2,147,483,647.

3.4 64-bit Integer

A 64 bit integer type must be provided called ‘**IR_Int64**’ with a domain of discrete values including the range -2,147,483, 647 to +2,147,483,647. If such a representation is not provided by the target machine’s instruction set architecture, a suitable data type may be implemented as a class defined in terms of two 32 bit integers.

3.5 Single Precision Floating Point

A 32 bit floating point type must be provided called **‘IR_FP32**’ as defined by single precision representations in IEEE Std. 754 or IEEE Std. 854.

3.6 Double Precision Floating Point

A 64 bit floating point type must be provided called **‘IR_FP64**’ as defined by double precision representations in IEEE Std. 754 or IEEE Std. 854.

3.7 IR_Kind

An enumerated type must be provided, called '**IR_Kind**', which uniquely identifies the IIR class associated with a particular object created from an IIR class. The enumeration may be implemented as a true enumerated type (preferred) or as an integer and set of constants (each named based on an enumeration label below). In either case, the type must include the following labels prior to any labels associated with completely new, instantiable IIR extension classes:

IR_DESIGN_FILE,
IR_COMMENT,
IR_IDENTIFIER,
IR_CHARACTER_LITERAL,
IR_STRING_LITERAL,
IR_BIT_STRING_LITERAL,
IR_INTEGER_LITERAL,
IR_INTEGER_LITERAL32,
IR_INTEGER_LITERAL64,
IR_FLOATING_POINT_LITERAL
IR_FLOATING_POINT_LITERAL32,
IR_FLOATING_POINT_LITERAL64,
IR_ASSOCIATION_ELEMENT_BY_EXPRESSION,
IR_ASSOCIATION_ELEMENT_BY_OTHERS,
IR_ASSOCIATION_ELEMENT_OPEN,
IR_BREAK_ELEMENT,
IR_CASE_STATEMENT_ALTERNATIVE_BY_EXPRESSION,
IR_CASE_STATEMENT_ALTERNATIVE_BY_CHOICES,
IR_CASE_STATEMENT_ALTERNATIVE_BY_OTHERS
IR_CHOICE,
IR_CONDITIONAL_WAVEFORM,
IR_COMPONENT_SPECIFICATION,
IR_BLOCK_CONFIGURATION,
IR_COMPONENT_CONFIGURATION,
IR_DESIGNATOR_EXPLICIT,
IR_DESIGNATOR_BY_OTHERS,
IR_DESIGNATOR_BY_ALL,
IR_ELSIF
IR_ENTITY_CLASS_ENTRY,
IR_SELECTED_WAVEFORM,
IR_SIMULTANEOUS_ALTERNATIVE_BY_EXPRESSION,
IR_SIMULTANEOUS_ALTERNATIVE_BY_CHOICES,
IR_SIMULTANEOUS_ALTERNATIVE_BY_OTHERS,
IR_SIMULTANEOUS_ELSIF
IR_WAVEFORM_ELEMENT,
IR_ASSOCIATION_LIST,
IR_ATTRIBUTE_SPECIFICATION_LIST,
IR_BREAK_LIST,
IR_CASE_ALTERNATIVE_LIST,
IR_CHOICE_LIST,
IR_COMMENT_LIST,
IR_CONCURRENT_STATEMENT_LIST,
IR_CONDITIONAL_WAVEFORM_LIST,
IR_CONFIGURATION_ITEM_LIST,

```
IR_DECLARATION_LIST,
IR_DESIGN_FILE_LIST,
IR_DESIGNATOR_LIST,
IR_ELEMENT_DECLARATION_LIST,
IR_NATURE_ELEMENT_DECLARATION_LIST,
IR_ENTITY_CLASS_ENTRY_LIST,
IR_ENUMERATION_LITERAL_LIST,
IR_GENERIC_LIST,
IR_INTERFACE_LIST,
IR_LIBRARY_UNIT_LIST,
IR_PORT_LIST,
IR_SELECTED_WAVEFORM_LIST,
IR_SEQUENTIAL_STATEMENT_LIST,
IR_SIMULTANEOUS_ALTERNATIVE_LIST,
IR_SIMULTANEOUS_STATEMENT_LIST,
IR_STATEMENT_LIST
IR_UNIT_LIST,
IR_WAVEFORM_LIST,
IR_ENUMERATION_TYPE_DEFINITION,
IR_ENUMERATION_SUBTYPE_DEFINITION,
IR_INTEGER_TYPE_DEFINITION,
IR_INTEGER_SUBTYPE_DEFINITION,
IR_FLOATING_TYPE_DEFINITION,
IR_FLOATING_SUBTYPE_DEFINITION,
IR_PHYSICAL_TYPE_DEFINITION,
IR_PHYSICAL_SUBTYPE_DEFINITION,
IR_RANGE_TYPE_DEFINITION,
IR_SCALAR_NATURE_DEFINITION,
IR_SCALAR_SUBNATURE_DEFINITION,
IR_ARRAY_TYPE_DEFINITION,
IR_ARRAY_SUBTYPE_DEFINITION,
IR_ARRAY_NATURE_DEFINITION,
IR_ARRAY_SUBNATURE_DEFINITION,
IR_RECORD_TYPE_DEFINITION,
IR_RECORD_SUBTYPE_DEFINITION,
IR_RECORD_NATURE_DEFINITION,
IR_RECORD_SUBNATURE_DEFINITION,
IR_PROTECTED_TYPE_DEFINITION,
IR_PROTECTED_TYPE_BODY,
IR_ACCESS_TYPE_DEFINITION,
IR_ACCESS_SUBTYPE_DEFINITION,
IR_FILE_TYPE_DEFINITION,
IR_SIGNATURE,
IR_FUNCTION_DECLARATION,
IR_PROCEDURE_DECLARATION,
IR_ELEMENT_DECLARATION,
IR_NATURE_ELEMENT_DECLARATION,
IR_ENUMERATION_LITERAL,
IR_TYPE_DECLARATION,
IR_SUBTYPE_DECLARATION,
IR_NATURE_DECLARATION,
IR_SUBNATURE_DECLARATION,
IR_CONSTANT_DECLARATION,
```

IR_FILE_DECLARATION,
IR_SIGNAL_DECLARATION,
IR_SHARED_VARIABLE_DECLARATION,
IR_VARIABLE_DECLARATION,
IR_TERMINAL_DECLARATION,
IR_FREE_QUANTITY_DECLARATION,
IR_ACROSS_QUANTITY_DECLARATION,
IR_THROUGH_QUANTITY_DECLARATION,
IR_SPECTRUM_SOURCE_QUANTITY_DECLARATION,
IR_NOISE_SOURCE_QUANTITY_DECLARATION,
IR_CONSTANT_INTERFACE_DECLARATION,
IR_FILE_INTERFACE_DECLARATION,
IR_SIGNAL_INTERFACE_DECLARATION,
IR_VARIABLE_INTERFACE_DECLARATION,
IR_TERMINAL_INTERFACE_DECLARATION,
IR_QUANTITY_INTERFACE_DECLARATION,
IR_ALIAS_DECLARATION,
IR_ATTRIBUTE_DECLARATION,
IR_COMPONENT_DECLARATION,
IR_GROUP_DECLARATION,
IR_GROUP_TEMPLATE_DECLARATION,
IR_LIBRARY_DECLARATION,
IR_ENTITY_DECLARATION,
IR_ARCHITECTURE_DECLARATION,
IR_PACKAGE_DECLARATION,
IR_PACKAGE_BODY_DECLARATION,
IR_CONFIGURATION_DECLARATION,
IR_PHYSICAL_UNIT,
IR_ATTRIBUTE_SPECIFICATION,
IR_CONFIGURATION_SPECIFICATION,
IR_DISCONNECTION_SPECIFICATION,
IR_LABEL,
IR_LIBRARY_CLAUSE,
IR_USE_CLAUSE,
IR_SIMPLE_NAME,
IR_SELECTED_NAME,
IR_SELECTED_NAME_BY_ALL,
IR_INDEXED_NAME,
IR_SLICE_NAME,
IR_USER_ATTRIBUTE,
IR_BASE_ATTRIBUTE,
IR_LEFT_ATTRIBUTE,
IR_RIGHT_ATTRIBUTE,
IR_LOW_ATTRIBUTE,
IR_HIGH_ATTRIBUTE,
IR_ASCENDING_ATTRIBUTE,
IR_IMAGE_ATTRIBUTE,
IR_VALUE_ATTRIBUTE,
IR_POS_ATTRIBUTE,
IR_VAL_ATTRIBUTE,
IR_SUCC_ATTRIBUTE,
IR_PRED_ATTRIBUTE,
IR_LEFT_OF_ATTRIBUTE,

```
IR_RIGHT_OF_ATTRIBUTE,
IR_RANGE_ATTRIBUTE,
IR_REVERSE_RANGE_ATTRIBUTE,
IR_LENGTH_ATTRIBUTE,
IR_DELAYED_ATTRIBUTE,
IR_STABLE_ATTRIBUTE,
IR_QUIET_ATTRIBUTE,
IR_TRANSACTION_ATTRIBUTE,
IR_EVENT_ATTRIBUTE,
IR_ACTIVE_ATTRIBUTE,
IR_LAST_EVENT_ATTRIBUTE,
IR_LAST_ACTIVE_ATTRIBUTE,
IR_LAST_VALUE_ATTRIBUTE,
IR_BEHAVIOR_ATTRIBUTE,
IR_STRUCTURE_ATTRIBUTE,
IR_DRIVING_ATTRIBUTE,
IR_DRIVING_VALUE_ATTRIBUTE,
IR_SIMPLE_NAME_ATTRIBUTE,
IR_INSTANCE_NAME_ATTRIBUTE,
IR_PATH_NAME_ATTRIBUTE,
IR_ACROSS_ATTRIBUTE,
IR_THROUGH_ATTRIBUTE,
IR_REFERENCE_ATTRIBUTE,
IR_CONTRIBUTION_ATTRIBUTE,
IR_TOLERANCE_ATTRIBUTE,
IR_DOT_ATTRIBUTE,
IR_INTEG_ATTRIBUTE,
IR_ABOVE_ATTRIBUTE,
IR_ZOH_ATTRIBUTE,
IR_LTF_ATTRIBUTE,
IR_ZTF_ATTRIBUTE,
IR_RAMP_ATTRIBUTE,
IR_SLEW_ATTRIBUTE,
IR_IDENTITY_OPERATOR,
IR_NEGATION_OPERATOR,
IR_ABSOLUTE_OPERATOR,
IR_NOT_OPERATOR,
IR_AND_OPERATOR,
IR_OR_OPERATOR,
IR_NAND_OPERATOR,
IR_NOR_OPERATOR,
IR_XOR_OPERATOR,
IR_XNOR_OPERATOR,
IR_EQUALITY_OPERATOR,
IR_INEQUALITY_OPERATOR,
IR_LESS_THAN_OPERATOR,
IR_LESS_THAN_OR_EQUAL_OPERATOR,
IR_GREATER_THAN_OPERATOR,
IR_GREATER_THAN_OR_EQUAL_OPERATOR,
IR_SLL_OPERATOR,
IR_SRL_OPERATOR,
IR_SLA_OPERATOR,
IR_SRA_OPERATOR,
```

IR_ROL_OPERATOR,
IR_ROR_OPERATOR,
IR_ADDITION_OPERATOR,
IR_SUBTRACTION_OPERATOR,
IR_CONCATENATION_OPERATOR,
IR_MULTIPLICATION_OPERATOR,
IR_DIVISION_OPERATOR,
IR_MODULUS_OPERATOR,
IR_REMAINDER_OPERATOR,
IR_EXPONENTIATION_OPERATOR,
IR_FUNCTION_CALL,
IR_PHYSICAL_LITERAL,
IR_AGGREGATE,
IR_OTHERS_INITIALIZATION,
IR_QUALIFIED_EXPRESSION,
IR_TYPE_CONVERSION,
IR_ALLOCATOR,
IR_WAIT_STATEMENT,
IR_ASSERTION_STATEMENT,
IR_REPORT_STATEMENT,
IR_SIGNAL_ASSIGNMENT_STATEMENT,
IR_VARIABLE_ASSIGNMENT_STATEMENT,
IR_PROCEDURE_CALL_STATEMENT,
IR_IF_STATEMENT,
IR_CASE_STATEMENT,
IR_FOR_LOOP_STATEMENT,
IR_WHILE_LOOP_STATEMENT,
IR_NEXT_STATEMENT,
IR_EXIT_STATEMENT,
IR_RETURN_STATEMENT,
IR_NULL_STATEMENT,
IR_BREAK_STATEMENT,
IR_BLOCK_STATEMENT,
IR_PROCESS_STATEMENT,
IR_SENSITIZED_PROCESS_STATEMENT,
IR_CONCURRENT_PROCEDURE_CALL_STATEMENT,
IR_CONCURRENT_ASSERTION_STATEMENT,
IR_CONCURRENT_CONDITIONAL_SIGNAL_ASSIGNMENT,
IR_CONCURRENT_SELECTED_SIGNAL_ASSIGNMENT,
IR_CONCURRENT_INSTANTIATION_STATEMENT,
IR_CONCURRENT_GENERATE_FOR_STATEMENT,
IR_CONCURRENT_GENERATE_IF_STATEMENT
IR_SIMPLE_SIMULTANEOUS_STATEMENT,
IR_CONCURRENT_BREAK_STATEMENT,
IR_SIMULTANEOUS_IF_STATEMENT,
IR_SIMULTANEOUS_CASE_STATEMENT,
IR_SIMULTANEOUS_PROCEDURAL_STATEMENT,
IR_SIMULTANEOUS_NULL_STATEMENT,
IR_PACKAGE_REGION,
IR_CONCURRENT_REGION,
IR_SIMULTANEOUS_REGION,
IR_SEQUENTIAL_REGION,
IR_PROCESS,

IR_SIGNAL,
IR_QUANTITY,
IR_DRIVER,
IR_EFFECTIVE_VALUE;
IR_NO_KIND

3.8 IR_SignalKind

An enumerated type must be provided, called '**IR_SignalKind**', which specifies various options associated with predefined IR_Signal and IR_SignalInterfaceDeclaration classes. The enumeration may be implemented as a true enumerated type or (preferred) as an integer and constant set. In either case, the type must include the following labels prior to any labels associated with completely new, instantiable IIR extension classes:

IR_NO_SIGNAL_KIND,
IR_REGISTER_KIND, and
IR_BUS_KIND

3.9 IR_Mode

An enumerated type must be provided, called '**IR_Mod**e', which specifies various options associated with predefined IR_InterfaceDeclaration classes. The enumeration may be implemented as a true enumerated type or (preferred) as an integer and constant set. In either case, the type must include the following labels prior to any labels associated with completely new, instantiable IIR extension classes:

IR_UNKNOWN_MODE,
IR_IN_MODE,,
IR_OUT_MODE,
IR_INOUT_MODE,
IR_BUFFER_MODE, and
IR_LINKAGE_MODE

3.10 IR_Pure

An enumerated type must be provided, called '**IR_Pure**', which specifies various options associated with predefined IR_FunctionDeclaration classes. The enumeration may be implemented as a true enumerated type or (preferred) as an integer and constant set. In either case, the type must include the following labels prior to any labels associated with completely new, instantiable IIR extension classes:

IR_UNKNOWN_PURE,
IR_PURE_FUNCTION,
IR_IMPURE_FUNCTION,
IR_PURE_PROCEDURAL,
IR_IMPURE_PROCEDURAL

3.11 IR_DelayMechanism

An enumerated type must be provided, called ‘**IR_DelayMechanism**’, which specifies various options associated with predefined signal assignment statement classes. The enumeration may be implemented as a true enumerated type or (preferred) as an integer and constant set. In either case, the type must include the following labels prior to any labels associated with completely new, instantiable IIR extension classes:

IR_UNKNOWN_DELAY,
IR_INERTIAL_DELAY,
IR_TRANSPORT_DELAY

3.12 IR_SourceLanguage

An enumerated type must be provided, called ‘IR_SourceLanguage’, which specifies the original source language in which an AIRE/CE fragment originated.

IR_VHDL87_SOURCE,
IR_VHDL93_SOURCE,
IR_VHDL98_SOURCE,
IR_VHDLAMS98_SOURCE,
IR_VERILOG95_SOURCE,
IR_VERILOG98_SOURCE

3.13 IIR Pointer

A variety of pointers to objects of IIR base class and each derived class must be provided. C++ provides for the assignment of a pointer from a derived class to a pointer to any class from which the more specific pointer is derived. Thus is is sufficient to utilize a single pointer, ‘**IIR***’ or more specific pointers.

CHAPTER 4

IIR Predefined Class Hierarchy

This chapter introduces the predefined IIR base class and those predefined classes derived from the IIR base class. A series of tables (Table 1 on page 42 to Table 11 on page 58) illustrate the IIR parent-child relationships in levels. Each level in the inheritance hierarchy is numbered beginning with the IIR base class (Level 1). Classes from which instances may created are denoted by *italics* in Table 1 on page 42 to Table 11 on page 58.

Subsequent chapters describe the predefined public methods and (abstract) data elements present in predefined IIR classes. All public, protected and private methods and data-elements added to those predefined in this specification by an IIR foundation implementation. must be prefaced by an underscore character ('_').

Between each predefined level in the IIR class hierarchy, one or more application-specific extension classes may be inserted. Application-specific methods may be inserted into the IIR class hierarchy as part of these extension classes. Furthermore, application-specific data elements may be inserted into any of the extension classes associated with instantiable (*italicized*) classes *except for lists*. For example, extension classes may add methods and data elements representing "temporary" information, new language features or even entirely new languages.

For documentation clarity, all such extension classes and declarators within these extensions are distinctly identified. The names of any intervening extension classes should assume the form **IIR** <Extension Designator> **'_'** <Specific class name>. For example, a synthesis application might interpose an extension class **IIRSyn_ProcessStatement** between **IIRBase_ProcessStatement** and **IIR_ProcessStatement**. Application specific methods and data elements must be prefaced by an underscore character, '_'.

There are several contexts in which the same source code fragment may be analyzed into one of several equivalent forms. A VHDL decimal literal may be analyzed into an IIR_Integer (general case) or IIR_Integer32 (special case of limited precision integer). A VHDL expression may be analyzed into a relational operator (general case) or into a specific function call to an implicit subprogram (specific case). A VHDL assertion statement may be analyzed into an equivalent process statement (general case) or into a concurrent assertion statement (special case). Since both general and specific representations are important for specific application domains, both representations are provided for in the IIR. A compliant implementation may choose to only provide for the general case, to provide for both the general and specific cases or (ideally) provide both general and specific representations with appropriate constructor methods.

4.1 IIR Class

The IIR class forms the base of the IIR design hierarchy, as shown in Table 1 on page 42. Application-specific extension classes may add additional methods (but not data elements) to the IIR base class. The IIR base class includes the IIRKind value associated with specific objects of derived IIR classes. Classes immediately derived from the IIR base class embody representation of the source location from which an IIR object originated.

TABLE 1. Class derivation hierarchy from IIR base class

Level 1 Base Class		Level 2 Derived Classes		Level 3 Derived Classes
IIR	E	*IIR_DesignFile*		
	E	*IIR_Comment*		
	E	IIR_Literal	E	...
	E	IIR_Tuple	E	...
	E	IIR_List	E	...
	E	IIR_TypeDefinition	E	...
	E	IIR_NatureDefinition		...
	E	IIR_Declaration	E	...
	E	IIR_Name	E	...
	E	IIR_Expression	E	...
	E	IIR_Statement	E	IIR_SequentialStatement
			E	IIR_ConcurrentStatement
			E	IIR_SimultaneousStatement
		IIR_Driver		
		IIR_EffectiveValue		

4.2 IIR_DesignFile Derived Class

IIR databases begin with one or more instances of the IIR_DesignFile class. Each instance of the IIR_DesignFile represents a source code file, schematic sheet or other designer-centric representation. Within the IIR_DesignFile, an IIR_DesignUnitList refers to zero or more design units which appear in the associated design file. An IIR_CommentList provides access to any comments stored when the design file was originally analyzed.

A static data element associated with the IIR_DesignFile class defines a list of design files known to a given IIR data base. An application may indirectly gain access to an entire IIR database via this list. Alternative entry points include lookup via textual names, resulting in one or more matching named entities (see Chapter 4 on page 41).

Additional, application-specific data elements and methods may be added to the IIRBase_DesignFile, resulting in an IIR_DesignFile class from which instances may be created. For example, this extension information may provide linkage information to external design representation databases and help to specify tools which should be used to display the original design representation.

4.3 IIR_Comment Derived Class

Each IIR_DesignFile includes an IIR_CommentList denoting a list of IIR_Comment objects. The IIR_Comment class represents the design file location and textual contents of a single comment.

Retention of comments during source code analysis is optional. A compliant IIR implementation or application may choose to remove comments in order to reduce IIR space consumption, obscure a human's ability to understand the design (for intellectual property protection) or other reasons.

If comments are retained, extended characters (e.g. unicode), capitalization, comment delimiters (e.g. VHDL's --) and spacing should generally be preserved in the comment object. This allows user interface applications to fully reconstruct source code fragments for user display. Examples of applications needing to reconstruct source code fragments include source level debuggers and profiling tools. Length-altering comment transformations should be undertaken with great care, if at all. Likewise, generation of IIR_Comments which did not appear in the original design representation should be undertaken with great care.

Application-specific information and methods may be added to IIRBase_Comment, resulting in an IIR_Comment class. Instances may then be created from the IIR_Comment class. For example, extensions may provide for comment encryption, translation, or hypertext linkages to other documents.

4.4 IIR_Literal Derived Class

IIR_Literal serves as the parent class for a variety of derived literal classes (see Table 2 on page 44). These literal classes represent explicit values within the IIR.

All literals classes may use a canonical representation. The same value may be represented by a common instance of a given (literal) class. Therefore, there are no methods for changing the value of an existing literal or unique source location values.

Predefined, static methods belonging to each derived literal class provide for obtaining the canonical representation of a particular value (get). These methods generally call a protected or private constructor method. Predefined methods provide the literal value in both binary and printable forms. Since formatting information, such as underscores and base, may be lost by a compliant implementation, printable access methods may need to address issues such as storage allocation, desired base, etc with additional parameters..

TABLE 2. **Class derivation hierarchy from IIRBase_Literal**

	Level 3 Derived Classes (only classes derived from IIRBase_Literal)		**Level 4 Derived Classes**		**Level 5 Derived Classes**
E	IIR_TextLiteral	E	*IIR_Identifier*		
		E	*IIR_StringLiteral*		
		E	*IIR_CharacterLiteral*		
		E	*IIR_BitStringLiteral*		
E	*IIIR_IntegerLiteral*				
E	*IIR_IntegerLiteral32*				
E	*IIR_IntegerLiteral64*				
E	*IIR_FloatingPointLiteral*				
E	*IIR_FloatingPointLiteral32*				
E	*IIR_FloatingPointLiteral64*				

Application-specific methods may be added to the IIRBase_Literal and any derived class, however data-elements may only be added to the derived classes of IIRBase_Literal, such as IIRBase_Integer32. Instances may then be created of the final, derived classes.

4.5 IIR_Tuple Derived Class

The IIR_Tuple class, shown in Table 3 on page 45, represent miscellaneous collections containing a predetermined, ordered set of items. Association elements, choices, and configuration items are further derived. Tuples are generally referenced from types, declarations, expressions or statements.

Extensions of the IIRBase_Tuple class provide for additional methods; extensions of the classes derived from IIRBase_Tuple provide for additional data elements and methods. For example, these extensions might provide for additional binding information or annotation of waveform entries.

TABLE 3. Class derivation hierarchy from IIR_Tuple

	Level 3 Derived Classes (only classes derived from IIR_Tuple)		**Level 4 Derived Classes**		**Level 5 Derived Classes**
E	IIR_AssociationElement	E	*IIR_AssociationElementExpression*		
E		E	*IIR_AssociationElemementByOthers*		
E		E	*IIR_AssociationElementOpen*		
E	*IIR_BreakElement*				
E	IIR_CaseStatementAlternative	E	*IIR_CaseStatementAlternativeByExpression*		
		E	*IIR_CaseStatementAlternativeByChoices*		
		E	*IIR_CaseStatementAlternativeByOthers*		
E	*IIR_Choice*				
E	*IIR_ConditionalWaveform*	E			
E	*IIR_ComponentSpecification*				
E	IIR_ConfigurationItem	E	*IIR_BlockConfiguration*		
		E	*IIR_ComponentConfiguration*		
	IIR_Designator	E	*IIR_DesignatorExplicit*		
		E	*IIR_DesignatorByOthers*		
		E	*IIR_DesignatorBy All*		
E	*IIR_EntityClassEntry*				
E	*IIR_SelectedWaveform*				
E	*IIR_SimultaneousElsif*				
E	*IIR_SimultaneousAlternative*	E	*IIR_SimultaneousAlternativeByExpression*		
		E	*IIR_SimultaneousAlternativeByChoices*		
		E	*IIR_SimultaneousAlternativeByAll*		
E	*IIR_WaveformElement*				

Predefined methods reference each of the items within tuples using method names derived from the applicable language reference manual grammar productions. These methods provide for both reading and writing individual items within a tuple.

4.6 IIR List Derived Class

Language grammars provide for ordered sets containing zero or more grammatical elements. Within the IIR, classes derived from IIR_List provide a convenient representation for such ordered sets. List classes provide pre-defined methods for identifying the first element in a list, determining the length of a list, and iterating forward or backward through a elements of a list. Table 4 on page 47 illustrates the classes derived directly from IIR_List. The IIR_InstantiationList has additional derived classes. .

TABLE 4. Class derivation hierarchy from IIR_List

Level 3 Derived Classes (only classes derived from IIR_List)		Level 4 Derived Classes		Level 5 Derived Classes
IIR_AssociationList				
IIR_AttributeSpecificationList				
IIR_BreakList				
IIR_CaseStatementAlternativeList				
IIR_ChoiceList				
IIR_CommentList				
IIR_ConcurrentStatementList				
IIR_ConditionalWaveformList				
IIR_ConfigurationItemList				
IIR_DeclarationList				
IIR_DesignFileList				
IIR_DesignatorList				
IIR_ElementDeclarationList				
IIR_EntityClassEntryList				
IIR_EnumerationLiteralList				
IIR_GenericList				
IIR_InterfaceList				
IIR_LibraryUnitList				
IIR_NatureElementDeclarationList				
IIR_PortList				
IIR_SelectedWaveformList				
IIR_SequentialStatementList				
IIR_SimultaneousStatementList				
IIR_SimultaneousAlternativeList				

TABLE 4. Class derivation hierarchy from IIR_List

IIR_UnitList				
IIR_WaveformList				

Most List classes are embedded directly into the public part of other IIR classes. This is in contrast to most other IIR classes, which are independently allocated and referenced by a pointer. Embedding helps to reduce memory consumption and allocation overhead. Since lists are directly instantiated, list extension classes *may not* add data elements.

4.7 IIR_TypeDefinition Derived Class

IIRBase_TypeDefinition classes represent a set (type) or subset (subtype) of values or types (signatures). Classes directly derived from IIR_TypeDefinition are shown in Table 5 on page 49. Type definitions may be referenced from type declarations, subtype declarations, subtype indications or implied declarations such as loop ranges.

Application-specific methods may be added to either IIRBase_TypeDefinition or any derived class, however additional data elements may only be added to the individual, derived type definition classes.

TABLE 5. Class derivation hierarchy from IIR_TypeDefinition

	Level 3 Derived Classes (only classes derived from IIR_TypeDefinition)		**Level 4 Derived Classes**		**Level 5 Derived Classes**
E	IIR_ScalarTypeDefinition	E	*IIR_EnumerationTypeDefinition*	E	*IIR_EnumerationSubtypeDefinition*
		E	*IIR_IntegerTypeDefinition*	E	*IIR_IntegerSubtypeDefinition*
		E	*IIR_FloatingTypeDefinition*	E	*IIR_FloatingSubtypeDefinition*
		E	*IIR_PhysicalTypeDefinition*	E	*IIR_PhysicalSubtypeDefinition*
		E	*IIR_RangeTypeDefinition*		
E	*IIR_ArrayTypeDefinition*	E	*IIR_ArraySubtypeDefinition*		
E	*IIR_RecordTypeDefinition*	E	*IIR_RecordSubtypeDefinition*		
E	*IIR_ProtectedTypeDeclaration*				
E	*IIR_ProtectedTypeBody*				
E	*IIR_AccessTypeDefinition*	E	*IIR_AccessSubtypeDefinition*		
E	*IIR_FileTypeDefinition*				
E	*IIR_Signature*				

4.8 IIR_NatureDefinition Derived Class

IIRBase_NatureDefinition classes represent a nature or subnature. Classes directly derived from IIR_NatureDefinition are shown in Table 6 on page 50.

TABLE 6. Class derivation hierarchy from IIR_NatureDefinition

	Level 3 Derived Classes (only classes derived from IIR_NatureDefinition)		Level 4 Derived Classes		Level 5 Derived Classes
E	IIR_ScalarNatureDefinition	E	IIR_ScalarSubnatureDefinition		
E	IIR_CompositeNatureDefinition	E	IIR_ArrayNatureDefinition		
		E	IIR_RecordNatureDefinition		IIR_RecordSubnatureDefinition

Application-specific methods may be added to either IIRBase_NatureDefinition or any derived class, however additional data elements may only be added to the individual, derived type definition classes

4.9 IIR Declaration Derived Class

The IIR_Declaration class and its derivative classes (see Table 7 on page 51) represent entities which can be referenced by names. Except for subprogram declarations, object declarations, interface declarations and library units, all of the classes directly derived from IIRBase_Declaration may be instantiated..

TABLE 7. Class derivation hierarchy from IIR_Declaration

	Level 3 Derived Classes (only classes derived from IIR_Declaration)		**Level 4 Derived Classes**		**Level 5 Derived Classes**		
E	IIR_SubprogramDeclaration	E	*IIR_FunctionDeclaration*				
		E	*IIR_ProcedureDeclaration*				
E	*IIR_EnumerationLiteral*						
E	*IIR_ElementDeclaration*						
E	*IIR_NatureElementDeclaration*						
E	*IIR_TypeDeclaration*						
E	*IIR_SubtypeDeclaration*						
E	*IIR_NatureDeclaration*						
E	*IIR_SubnatureDeclaration*						
E	IIR_ObjectDeclaration	E	*IIR_TypedObjectDeclaration*	E	*IIR_ConstantDeclaration*		
				E	*IIR_VariableDeclaration*		
				E	*IIR_SharedVariableDeclaration*		
				E	*IIR_SignalDeclaration*		
				E	*IIR_FileDeclaration*		
				E	IIR_QuantityDeclaration	E	*IIR_FreeQuantityDeclaration*
				E		*E*	*IIR_AcrossQuantityDeclaration*
				E		*E*	*IIR_NoiseSourceQuantityDeclaration*
				E		*E*	*IIR_SpectrumSourceQuantity Declaration*
				E		*E*	*IIR_ThroughQuantityDeclaration*
			IIR_NaturedObjectDeclaration	E	*IIR_TerminalDeclaration*		
E	IIR_InterfaceDeclaration	E	*IIR_TypedInterfaceDeclaration*	E	*IIR_ConstantInterfaceDeclaration*		

TABLE 7. Class derivation hierarchy from IIR_Declaration

		E		E	*IIR_VariableInterfaceDeclaration*		
		E		E	*IIR_SignalInterfaceDeclaration*		
		E		E	*IIR_FileInterfaceDeclaration*		
		E		E	*IIR_TerminalInterfaceDeclaration*		
		E	*IIR_NaturedInterfaceDeclaration*	E	*IIR_QuantityInterfaceDeclaration*		
E	*IIR_AliasDeclaration*						
E	*IIR_AttributeDeclaration*						
E	*IIR_ComponentDeclaration*						
E	*IIR_GroupDeclaration*						
E	*IIR_GroupTemplateDeclaration*						
E	*IIR_LibraryDeclaration*						
E	IIR_LibraryUnit	E	*IIR_EntityDeclaration*				
		E	*IIR_ArchitectureDeclaration*				
		E	*IIR_PackageDeclaration*				
		E	*IIR_PackageBodyDeclaration*				
		E	*IIR_ConfigurationDeclaration*				
E	*IIR_PhysicalUnit*						
E	*IIR_AttributeSpecification*						
E	*IIR_ConfigurationSpecification*						
E	*IIR_DisconnectionSpecification*						
E	*IIR_Label*						
E	*IIR_LibraryClause*						
E	*IIR_UseClause*						

4.10 IIR Name Derived Class

Names denote declared entities, objects denoted by access values, sub-elements of composite objects, sub-elements of composite values, slices of composite objects, slices of composite values and attributes of any named entity.

Table 8 on page 53 illustrates the classes derived from IIR_Name. Both attributes and entity names are further decomposed into additional derived, instantiable classes. The remaining derived base name classes are directly instantiable. Additional methods and data elements may be added to IIRBase_Name and classes derived from IIR_Name.

TABLE 8. Class hierarchy derived from IIR_Name

	Level 3 Derived Classes (only classes derived from IIR_Name)		**Level 4 Derived Classes**		**Level 5 Derived Classes**
E	*IIR_SimpleName*				
E	*IIR_SelectedName*				
E	*IIR_SelectedNameByAll*				
E	*IIR_IndexedName*				
E	*IIR_SliceName*				
E	IIR_Attribute	E	*IIR_UserAttribute*		
		E	*IIR_BaseAttriBute*		
		E	*IIR_LeftAttribute*		
		E	*IIR_RightAttribute*		
		E	*IIR_LowAttribute*		
		E	*IIR_HighAttribute*		
		E	*IIR_AscendingAttribute*		
		E	*IIR_ImageAttribute*		
		E	*IIR_ValueAttribute*		
		E	*IIR_PosAttribute*		
		E	*IIR_ValAttribute*		
		E	*IIR_SuccAttribute*		
		E	*IIR_PredAttribute*		
		E	*IIR_LeftOfAttribute*		
		E	*IIR_RightOfAttribute*		
		E	*IIR_RangeAttribute*		
		E	*IIR_ReverseRangeAttribute*		
		E	*IIR_LengthAttribute*		

TABLE 8. Class hierarchy derived from IIR_Name

		E	*IIR_DelayedAttribute*		
		E	*IIR_StableAttribute*		
		E	*IIR_QuietAttribute*		
		E	*IIR_TransactionAttribute*		
		E	*IIR_AscendingAttribute*		
		E	*IIR_EventAttribute*		
		E	*IIR_ActiveAttribute*		
		E	*IIR_LastEventAttribute*		
		E	*IIR_LastActiveAttribute*		
		E	*IIR_LastValueAttribute*		
		E	*IIR_DrivingAttribute*		
		E	*IIR_DrivingValueAttribute*		
		E	*IIR_SimpleNameAttribute*		
		E	*IIR_InstanceNameAttribute*		
		E	*IIR_PathNameAttribute*		
		E	*IIR_AcrossAttribute*		
		E	*IIR_ThroughAttribute*		
		E	*IIR_ReferenceAttribute*		
		E	*IIR_ContributionAttribute*		
		E	*IIR_ToleranceAttribute*		
		E	*IIR_DotAttribute*		
		E	*IIR_IntegAttribute*		
		E	*IIR_AboveAttribute*		
		E	*IIR_ZOHAttribute*		
		E	*IIR_LTFAttribute*		
		E	*IIR_ZTFAttribute*		

4.11 IIR_Expression Derived Class

The IIRBase_Expression class and its derivatives, shown in Table 9 on page 55 define the computation of a value. The operator classes are further derived; all other expression classes may be directly instantiated.

Application-specific data elements or methods may be added to the IIRBase_Expression extension class or any derived extension class. For example, these extensions may provide additional operator implementation information.

TABLE 9. Class derivation hierarchy from IIRBase_Expression

	Level 3 Derived Classes (only classes derived from IIR_Expression)		Level 4 Derived Classes		Level 5 Derived Classes
E	IIR_MonadicOperator	E	*IIR_IdentityOperator*		
		E	*IIR_NegationOperator*		
		E	*IIR_AbsoluteOperator*		
		E	*IIR_NotOperator*		
E	IIR_DyadicOperator	E	*IIR_AndOperator*		
		E	*IIR_OrOperator*		
		E	*IIR_NandOperator*		
		E	*IIR_NorOperator*		
		E	*IIR_XorOperator*		
		E	*IIR_XnorOperator*		
		E	*IIR_EqualityOperator*		
		E	*IIR_InequalityOperator*		
		E	*IIR_LessThanOperator*		
		E	*IIR_LessThanOrEqualOperator*		
		E	*IIR_GreaterThanOperator*		
		E	*IIR_GreaterThanOrEqualOperator*		
		E	*IIR_SLLOperator*		
		E	*IIR_SRLOperator*		
		E	*IIR_SLAOperator*		
		E	*IIR_SRAOperator*		
		E	*IIR_ROLOperator*		
		E	*IIR_ROROperator*		
		E	*IIR_AdditionOperator*		
		E	*IIR_SubtractionOperator*		

TABLE 9. Class derivation hierarchy from IIRBase_Expression

		E	*IIR_ConcatentationOperator*		
		E	*IIR_MultiplicationOperator*		
		E	*IIR_DivisionOperator*		
		E	*IIR_ModulusOperator*		
		E	*IIR_RemainderOperator*		
		E	*IIR_ExponentiationOperator*		
E	*IIR_FunctionCall*				
E	*IIR_PhysicalLiteral*				
E	*IIR_Aggregate*				
E	*IIR_OthersInitialization*				
E	*IIR_QualifiedExpression*				
E	*IIR_TypeConversion*				
E	*IIR_Allocator*				

The operator derived classes represent the most general case of predefined and overloaded operators. Type and visibility analysis allows further specific of operators using function call to a foreign or implied function declaration. Compliant implementations may use either notations, support both notations or (ideally) both notations with inter-conversion constructors.

4.12 IIR Sequential Statement Derived Class

The IIRBase_SequentialStatement class and its derivatives (see Table 10 on page 57) represent the set of sequentially executed statements within a process. Most such statements have two to three arguments and single, default subsequent statement.

The IIR_SequentialStatement and its derivatives may be extended by both data elements and methods using the extension class layers. For example, these extensions may provide additional analytic or implementation information.

TABLE 10. Class hierarchy derived from IIR_SequentialStatement class

	Level 4 Derived Classes (only classes derived from IIR_SequentialStatement)		**Level 5 Derived Classes**		**Level 6 Derived Classes**
E	*IIR_WaitStatement*				
E	*IIR_AssertionStatement*				
E	*IIR_ReportStatement*				
E	*IIR_SignalAssignmentStatement*				
E	*IIR_VariableAssignmentStatement*				
E	*IIR_ProcedureCallStatement*				
E	*IIR_IfStatement*				
E	*IIR_CaseStatement*				
E	*IIR_ForLoopStatement*				
E	*IIR_WhileLoopStatement*				
E	*IIR_NextStatement*				
E	*IIR_ExitStatement*				
E	*IIR_ReturnStatement*				
E	*IIR_NullStatement*				
E	*IIR_BreakStatement*				

4.13 IIR Concurrent Statement Derived Class

The IIRBase_ConcurrentStatement class and its derivatives (see Table 11 on page 58) represent block and process structure..

TABLE 11. Class hierarchy derived from IIR_ConcurrentStatement

	Level 4 Derived Classes (only classes derived from IIR_ConcurrentStatement)		**Level 5 Derived Classes**		**Level 6 Derived Classes**
E	*IIR_BlockStatement*				
E	*IIR_ProcessStatement*	E	*IIR_SensitizedProcessStatement*		
E	*IIR_ConcurrentProcedureCallStatement*				
E	*IIR_ConcurrentAssertionStatement*				
E	*IIR_ConcurrentConditionalSignalStatement*				
E	*IIR_ConcurrentSelectedSignalStatement*				
E	*IIR_ComponentInstantiationStatement*				
E	*IIR_ConcurrentGenerateForStatement*				
E	*IIR_ConcurrentGenerateIfStatement*				

4.14 IIR Simultaneous Statement Derived Class

The IIRBase_SimultaneousStatement class and its derivatives (see Table 12 on page 59) represent simultaneous statements used by VHDL-AMS.

TABLE 12. Class hierarchy derived from IIR_SimultaneousStatement

	Level 4 Derived Classes (only classes derived from IIR_SimultaneousStatement)		Level 5 Derived Classes		Level 6 Derived Classes
E	*IIR_SimpleSimultaneousStatement*				
E	*IIR_ConcurrentBreakStatement*				
E	*IIR_SimultaneousIfStatement*				
E	*IIR_SimultaneousCaseStatement*				
E	*IIR_SimultaneousProceduralStatement*				
E	*IIR_SimultaneousNullStatement*				

CHAPTER 5

IIR Base Class

5.1 IIR

5.1.1 Derived Class Description

IIR is the base class from which all other IIR classes are descended. Objects cannot be directly created from the IIR class. Pointers to an object having any class derived from IIR may be assigned to a pointer to IIR. Conversely, assigning a pointer to IIR to any more specific IIR pointer requires a cast. The IIR class hierarchy has been designed to minimize the need for casting.

5.1.2 Properties

TABLE 13. IIR Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR**
Predefined child classes	IIR_DesignFile IIR_Comment IIR_Literal IIR_Tuple IIR_List IIR_TypeDefinition IIR_NatureDefinition IIR_Declaration IIR_Name IIR_Expression IIR_Statement
Instantiation?	Indirectly via any of the derived classes of IIR
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR class
Public data elements	None

5.1.3 Predefined Public Methods (C++)

All of the following methods must be applied to a valid object having a class derived from IIR. All of the following methods are atomic.

5.1.3.1 IR_Kind Methods

An IIR object's IR_Kind is determined at the time it is constructed and may not be changed.

```
IR_Kind
    get_kind();
```

A static method provides a character string corresponding to the enumerated value of each IR_Kind.

```
static IR_Char*
    get_kind_text(IR_Kind         kind);
```

5.1.3.2 Source Location Methods

Source information allows select IIR objects to be associated directly with the design representation visible to an end-user of the application. The IIR classes which actually retain source location information depends on the specific IIR foundation implementation. Source location methods for objects which have no specific information refer to an 'unknown' file with offset and line number '-1'.

```
void
    set_file_name(IIR_Identifier*         file_name);
IIR_Identifier*
    get_file_name();
void
    set_character_offset(IR_Int32         character_offset);
IR_Int32
    get_character_offset();
void
    set_line_number(IR_Int32              line_number);
IR_Int32
    get_line_number();
void
    set_column_number(IR_Int32            column_number);
IR_Int32
    get_column_number();
void
    set_sheet_name(IIR_Identifier*        sheet_name);
IIR_Identifier*
    get_sheet_name();
void
    set_x_coordinate(IR_Int32             x_coordinate);
IR_Int32
    get_x_coordinate();
void
    set_y_coordinate(IR_Int32             y_coordinate);
IR_Int32
    get_y_coordinate();
```

5.2 IIR_Statement

5.2.1 Derived Class Description

The predefined **IIR_Statement** classes specify individual sequential statements within a process or subprogram.

5.2.2 Properties

TABLE 14. IIR_SequentialStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR**
Predefined child classes	**IIR_SequentialStatement** **IIR_ConcurrentStatement** **IIR_SimultaneousStatement**
Instantiation?	Indirectly via any of the derived classes of IIR_Statement
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_Statement class
Predefined public data elements	None

5.2.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_Statement. All of the following methods are atomic.

5.2.3.1 Label Methods

Label methods associate an IIR_Label (declarator) with a sequential statement object. Access to the label of a statement which lacks a label may return either NIL or an automatically generated label. Setting the label value to NIL disassociates any previously defined label. All automatically generated labels must be unique within the enclosing declarative region.

void

```
    set_label(     IIR_Label*                      label);
IIR_Label*
    get_label();
```

CHAPTER 6

IIR_DesignFile & IIR_Comment Derived Classes

6.1 IIR_DesignFile

6.1.1 Derived Class Description

The predefined **IIR_DesignFile** class represents the textual contents of a design file. These contents may include one or more IIR_LibraryUnits and/or one or more IIR_Comments.

6.1.2 Properties

TABLE 15. IIR_DesignFile Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_DESIGN_FILE**
Parent class	**IIR**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes

6.1.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_DesignFile. All of the following methods are atomic.

6.1.3.1 Constructor Method

The constructor method initializes an IIR_DesignFile object with an undefined file name file_name.

```
IIR_DesignFile();
```

6.1.3.2 Name Method

The name method associates an IIR_Identifier with the file, denoting the file name and potentially a path to the file.

```
void
    set_name(IIR_Identifier*              name);
IIR_Identifier*
    get_name();
```

6.1.3.3 Source Language Methods

The source language method identifies the source language associated with this design file.

```
void
    set_source_language(IR_SourceLanguage        source_language);
IR_SourceLanguage
    get_source_language();
```

6.1.3.4 Destructor Method

The destructor method deletes storage associated with the design file’s comments (if present) and library units (if present) before deallocating the design file object itself.

```
    ~IIR_DesignFile();
```

6.1.4 Predefined Public Data Elements

IIR_DesignFiles have the following predefined static and per-instance data elements. The static list of design files provides access to the list of design files contained within an IIR database. The per-instance data elements provide access to the list of comments and library units contained within a single design file.

```
static IIR_DesignFileList         design_files;

IIR_CommentList                   comments;
IIR_LibraryUnitList               library_units;
```

6.2 IIR_Comment

6.2.1 Derived Class Description

The predefined **IIR_Comment** class represents a single, contiguous comment within the original source code.

6.2.2 Properties

TABLE 16. IIR_Comment Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_COMMENT**
Parent class	**IIR**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

6.2.3 Predefined Public Methods (C++)

Except for the get method, all of the following methods must be applied to a valid object having a class derived from IIR_Comment. All of the following methods are atomic.

6.2.3.1 Get Method

The static get method returns a pointer to a comment object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms.

```
static IIR_Comment*
    get(    IIR_Char*               text,
            IR_Int32                text_length);
```

6.2.3.2 Value & Length Methods

The value method returns a previously allocated array of characters; the caller must *not* deallocate the storage returned. Note that the array of characters may not be null terminated; the length does not include any such reserved termination character.

```
IR_Char*
    get_text();
IR_Int32
    get_text_length();
```

6.2.3.3 Element Access Methods

Element access methods provide element-by-element access to the elements of a comment. It returns Character values from a zero-origined array.

```
void
    set_element(   IR_Int32             subscript,
                   IR_Char              value);

IIR_Char
    operator[] (   IR_Int32             subscript);
```

6.2.3.4 Release Method

The release method release the IIR_Comment previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

CHAPTER 7

IIR_Literal Derived Classes

Literals represent values explicit in the source code. This Chapter specifies the properties, predefined public methods, predefined functions and predefined data elements used to represent the literal derived classes shown in Table 17 on page 69. All derivative classes of IIR_Literal are dynamically and individually allocated objects of IIR_LiteralDefinition..

TABLE 17. Class derivation hierarchy from IIRBase_Literal

	Level 3 Derived Classes (only classes derived from IIRBase_Literal)		**Level 4 Derived Classes**		**Level 5 Derived Classes**
E	IIR_TextLiteral	D	IIR_Identifier		
		E	*IIR_CharacterLiteral*		
		E	*IIR_StringLiteral*		
		E	*IIR_BitStringLiteral*		
E	*IIR_IntegerLiteral*	E		E	
E	*IIR_IntegerLiteral32*	E		E	
E	*IIR_IntegerLiteral64*	E		E	
E	*IIR_FloatingPointLiteral*	E		E	
E	*IIR_FloatingPointLiteral32*	E		E	
E	*IIR_FloatingPointLiteral64*	E		E	

7.1 IIR_Literal

7.1.1 Derived Class Description

The predefined **IIR_Litera**l class, derived from IIR, is the parent class for all literal values.

7.1.2 Properties

TABLE 18. **IIR_Literal Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR**
Predefined child classes	**IIR_TextLiteral** **IIR_IntegerLiteral** **IIR_IntegerLiteral32** **IIR_IntegerLiteral64** **IIR_FloatingPointLiteral** **IIR_FloatingPointLiteral32** **IIR_FloatingPointLiteral64**
Instantiation?	Indirectly via any of the derived classes of IIR_Literal
Application-specific data elements	Except for IIR_TextLiteral, via extension classes associated with specific derived classes of the IIR_Literal class
Public data elements	None

7.1.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_Literal. All of the following methods are atomic.

7.2 IIR_TextLiteral

7.2.1 Derived Class Description

The predefined **IIR_TextLitera**l class, derived from IIR, is the parent class for all explicit literal values consisting of an array of zero or more characters.

7.2.2 Properties

TABLE 19. IIR_TextLiteral Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Literal**
Predefined child classes	**IIR_Identifier** **IIR_CharacterLiteral** **IIR_StringLiteral** **IIR_BitStringLiteral**
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_TextLiteral class
Public data elements	None

7.2.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_TextLiteral. All of the following methods are atomic.

7.3 IIR_Identifier

7.3.1 Derived Class Description

The predefined **IIR_Identifier** class represents identifiers, VHDL extended identifiers and quoted VHDL operator names. Note that identifiers, extended identifiers, and quoted operator names are only distinguished by their textual value, not by the IIR class hierarchy.

7.3.2 Properties

TABLE 20. IIR_Identifier Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_IDENTIFIER**
Parent class	**IIR_TextLiteral**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

7.3.3 Predefined Public Methods (C++)

All of the following methods must be applied to a valid object having a class derived from IIR_Identifier. All of the following methods are atomic.

7.3.3.1 Get Method

This method allows either construction of a unique IIR_Identifier or re-use of an existing identifier having the specified text and length.

```
IIR_Identifier*
    get(    IR_Char*       text,
            IR_Int32       length);
```

7.3.3.2 Text Methods

The text methods refer to a previously allocated array of characters; the caller must *not* deallocate the storage returned. Note that the array of characters may not be null terminated; the length does not include any such reserved termination character. Once an identifier has been acquired via the get method above, it's text may not be altered. The text length includes a leading and trailing double quotes in the case of quoted VHDL operator names.

```
IR_Char*
    get_text();
IR_Int32
    get_text_length();
```

7.3.3.3 Destructor Method

The release method release the IIR_Identifier previously constructed.

```
    ~IIR_Identifier();
```

7.4 IIR_CharacterLiteral

7.4.1 Derived Class Description

The predefined **IIR_CharacterLiteral** class represents character literals defined by ISO Std. 8859-1.

7.4.2 Properties

TABLE 21. **IIR_CharacterLiteral Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_CHARACTER_LITERAL**
Parent class	**IIR_TextLiteral**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

7.4.3 Predefined Public Methods (C++)

The IIR_CharacterLiteral class's methods other than the constructor must be applied to a valid object of IIR_CharacterLiteral class. All of the following methods are atomic.

7.4.3.1 Get Method

The get method returns a pointer to a literal having a character value specified by ISO Std. 8859-1. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms

```
static IIR_CharacterLiteral*
    get(            IIR_Char                         character);
```

7.4.3.2 Value Method

The value method returns an ISO Std. 8859-1 representation of the character.

```
IIR_Char
    get_text();
```

7.4.3.3 Destructor Method

The destructor method deletes the character literal object itself.

```
~IIR_CharacterLiteral();
```

7.5 IIR_StringLiteral

7.5.1 Derived Class Description

The predefined **IIR_StringLiteral** class represents an array of zero or more character literals defined by ISO Std. 8859-1.

7.5.2 Properties

TABLE 22. IIR_StringLiteral Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_STRING_LITERAL**
Parent class	**IIR_TextLiteral**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

7.5.3 Predefined Public Methods

The IIR_StringLiteral class's methods other than get must be applied to a valid object of IIR_StringLiteral class. All of the following methods are atomic.

7.5.3.1 Get Method

The get method returns a pointer to a string literal object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms.

```
static IIR_StringLiteral*
    get_value(    IR_Char*            value,
                  IIR_Int32           length);
```

7.5.3.2 Text Methods

The text methosd returns a previously allocated array of characters; the caller must *not* deallocate the storage returned. Note that the array of characters may not be null terminated; the length does not include any such reserved termination character.

```
IR_Char*
    get_text();
IR_Int32
    get_text_length();
```

7.5.3.3 Element Access Methods

Element access methods provide element-by-element access to the elements of a string literal. It returns character values from a zero-origined array.

```
IR_Char
    operator[]     (IR_Int32      subscript);

void
    set_element(   IR_Int32       subscript,
                   IR_Char        value);
```

7.5.3.4 Release Method

The release method release the IIR_StringLiteral previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void

    release();
```

7.6 IIR_BitStringLiteral

7.6.1 Derived Class Description

The **IIR_BitStringLiteral** represents an array of zero or more literals having either character literal value ‘0’ or ‘1’.

7.6.2 Properties

TABLE 23. IIR_BitStringLiteral Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IIR_BIT_STRING_LITERAL**
Parent class	**IIR_TextLiteral**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

7.6.3 Predefined Public Methods

The IIR_BitStringLiteral class’s methods other than the constructor must be applied to a valid object of IIR_BitStringLiteral class. All of the following methods are atomic.

7.6.3.1 Get Method

The get method returns a pointer to a bit string literal object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms. The value used by get must be an array of characters having value ‘0’ or ‘1’.

```
static IIR_BitStringLiteral*
    get(    IR_Char*        value,
            IR_Int32        length);
```

7.6.3.2 Text Methods

The text methods returns a uniquely allocated array of characters; the caller must deallocate the storage returned when the caller is done with the storage. Note that the array of characters may not be null terminated; the length does not include any such reserved termination character.

```
IR_Char*
    get_text();
IR_Int32
    get_text_length();
```

7.6.3.3 Element Access Methods

Element access methods provide element-by-element access to the elements of a bit string literal. It returns numeric 0 or 1 values from a zero-origined array.

```
IIR_Int32
    operator[] (   IR_Int32            subscript);
void
    set_element(   IR_Int32            subscript,
                   IR_Int32            value);
```

7.6.3.4 Release Method

The release method release the IIR_BitStringLiteral previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void

    release();
```

7.7 IIR_IntegerLiteral

7.7.1 Derived Class Description

The predefined **IIR_IntegerLiteral** class is the most general representation of an integer literal. It is capable of representing *any* integer literal value falling within the limitations of a specific IIR foundation implementation.

7.7.2 Properties

TABLE 24. IIR_Integer Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_INTEGER_LITERAL**
Parent class	**IIR_Literal**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

7.7.3 Predefined Public Methods

The IIR_Integer class's methods other than the constructor must be applied to a valid object of IIR_Integer class. All of the following methods are atomic.

7.7.3.1 Get Method

The get method returns a pointer to a 32 bit integer literal object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms.

```
static IIR_IntegerLiteral*
    get(    IR_Int32                base,
            IR_Char*                mantissa,
            IR_Int32                mantissa_length,
            IR_Char*                exponent,
            IR_Int32                exponent_length);
```

7.7.3.2 Value Methods

TBD

7.7.3.3 Print Method

The print method converts the integer value into a character string of the specified length. Depending on the IIR implementation, the caller may or may not be responsible for deallocating storage used for the output string.

```
IR_Char*
    print(  IR_Int32&                    length);
```

7.7.3.4 Release Method

The release method releases the IIR_IntegerLiteral previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

7.8 IIR_IntegerLiteral32

7.8.1 Derived Class Description

The predefined **IIR_IntegerLiteral32** class is an integer literal class capable of representing any literal value within the range covered by a 32 bit signed, two's complement representation.

7.8.2 Properties

TABLE 25. IIR_IntegerLiteral32 Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A,
IR_Kind enumeration value	**IR_INTEGER_LITERAL32**
Parent class	**IIR_Literal**
Predefined child classes	None
Instantiation?	Dynamically instantiated via new
Application-specific data elements	Via extension class
Public data elements	None

7.8.3 Predefined Public Methods

The IIR_Integer32 class's methods other than the constructor must be applied to a valid object of IIR_Integer32 class. All of the following methods are atomic.

7.8.3.1 Get Method

The get method returns a pointer to a 32 bit integer literal object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms.

```
static IIR_IntegerLiteral32*
    get(    IIR_Int32       v);
```

7.8.3.2 Value Methods

The value methods reference the integer as a 32 bit signed integer.

```
IR_Int32
    value();
```

7.8.3.3 Release Method

The release method releases the IIR_IntegerLiteral32 literal previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

7.9 IIR_IntegerLiteral64

7.9.1 Derived Class Description

The prdefined **IIR_IntegerLiteral64** class is an integer literal class capable of representing any literal value within the range covered by a 64 bit signed, two's complement representation.

7.9.2 Properties

TABLE 26. **IIR_IntegerLiteral64 Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IIR_INTEGER_LITERAL64**
Parent class	**IIR_Literal**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

7.9.3 Predefined Public Methods

The IIR_Integer64 class's methods other than the constructor must be applied to a valid object of IIR_Integer64 class. All of the following methods are atomic.

7.9.3.1 Get Method

The get method returns a pointer to a 64 bit integer literal object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms.

```
static IIR_IntegerLiteral64*
    get(   IR_Int64      v);
```

7.9.3.2 Value Methods

The value methods reference the integer as a 64 bit signed integer.

```
IR_Int64
    value();
```

7.9.3.3 Release Method

The release method releases the IIR_Integer64 literal previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
release();
```

7.10 IIR_FloatingPointLiteral

7.10.1 Derived Class Description

IIR_FloatingPointLiteral is the most general representation of a floating point literal. It is capable of representing any floating point literal value within the implementation-defined limitations of a specific IIR foundation.

7.10.2 Properties

TABLE 27. IIR_FloatingPointLiteral Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_FLOATING_POINT_LITERAL**
Parent class	IIR_LITERAL
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

7.10.3 Predefined Public Methods

The IIR_FloatingPointLiteral class's methods other than the get method must be applied to a valid object of IIR_FloatingPointLiteral class. All of the following methods are atomic.

7.10.3.1 Get Method

The get method returns a pointer to a generic floating point literal object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms. The decomposed character string must be a valid VHDL floating point literal.

```
static IIR_FloatingPointLiteral*
    get(    IR_Int32            base,
            IR_Char*            mantissa,
            IR_Int32            mantissa_length,
            IR_Char*            exponent,
            IR_Int32            exponent_length);
```

7.10.3.2 Value Methods

```
TBD
```

7.10.3.3 Print Method

The print method converts the floating point value into a character string of the specified length. Depending on the IIR implementation, the caller may or may not be responsible for deallocating storage used for the output string.

```
IR_Char*
    print(          IR_Int32&               length);
```

7.10.3.4 Release Method

The release method releases the IIR_FloatingPointLiteral previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

7.11 IIR_FloatingPointLiteral32

7.11.1 Derived Class Description

The predefined **IIR_FloatingPointLiteral32** is a floating point literal class capable of representing any literal value within the range covered by an IEEE single precision representation.

7.11.2 Properties

TABLE 28. IIR_FloatingPointLiteral32 Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_FLOATING_POINT_LITERAL32**
Parent class	**IIR_Literal**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

7.11.3 Predefined Public Methods

The IIR_FloatingPointLiteral32 class's methods other than the get method must be applied to a valid object of IIR_FloatingPointLiteral32 class. All of the following methods are atomic.

7.11.3.1 Get Method

The get method returns a pointer to a 32 bit floating point literal object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms.

```
static IIR_FloatingPointLiteral32*
    get_value(     IR_FP32        value);
```

7.11.3.2 Value Methods

The value methods reference the literal as a single precision IEEE floating point value.

```
IR_FP32
    get_value();
```

7.11.3.3 Release Method

The release method releases the IIR_FloatingPointLiteral32 previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

7.12 IIR_FloatingPointLiteral64

7.12.1 Derived Class Description

The predefined **IIR_FloatingPointLiteral64** class is capable of representing any literal value within the range covered by an IEEE double precision representation.

7.12.2 Properties

TABLE 29. IIR_FloatingPointLiteral64 Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_FLOATING_POINT_LITERAL64**
Parent class	**IIR_Literal**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

7.12.3 Predefined Public Methods

The IIR_FloatingPointLiteral64 class's methods other than the constructor must be applied to a valid object of IIR_FloatingPointLiteral64 class. All of the following methods are atomic.

7.12.3.1 Get Method

The get method returns a pointer to a 64 bit floating point literal object having the specified value. Note that the get method can be implemented via a protected or private call to new or other via other allocation mechanisms.

```
static IIR_FloatingPointLiteral64*
    get_value(    IR_FP64              value);
```

7.12.3.2 Value Method

The value methods reference the literal as a double precision IEEE floating point value.

```
IR_FP64
    value();
```

7.12.3.3 Release Method

The release method releases a IIR_FloatingPointLiteral64 object previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

CHAPTER 8

IIR Tuple Derived Classes

Classes derived from IIR Tuple represent specific collections of information related to a particular aspect of the design being represented. For a specific class derived from IIR Tuple, the set of information is finite and well defined. Table 30 on page 93 illustrates the design hierarchy derived from IIR_Tuple..

TABLE 30. Class derivation hierarchy from IIR_Tuple

	Level 3 Derived Classes (only classes derived from IIR_Tuple)	E	Level 4 Derived Classes	E	Level 5 Derived Classes
E	IIR_AssociationElement	E	*IIR_AssociationElementByExpression*		
E		E	*IIR_AssociationElementByOthers*		
		E	*IIR_AssociationElementOpen*		
E	*IIR_BreakElement*				
E	IIR_CaseStatementAlternative	E	*IIR_CaseStatementAlternativeByExpression*		
		E	*IIR_CaseStatementAlternativeByChoices*		
		E	*IIR_CaseStatementAlternativeByOthers*		
E	*IIR_Choice*				
E	*IIR_ComponentSpecification*				
E	*IIR_ConditionalWaveform*	E			
E	IIR_ConfigurationItem	E	*IIR_BlockConfiguration*		
		E	*IIR_ComponentConfiguration*		
	IIR_Designator	E	*IIR_DesignatorExplicit*		
		E	*IIR_DesignatorByOthers*		
		E	*IIR_DesignatorByAll*		
E	*IIR_Elsif*				
E	*IIR_EntityClassEntry*				
E	*IIR_SelectedWaveform*				

TABLE 30. **Class derivation hierarchy from IIR_Tuple**

E	*IIR_SimultaneousAlternative*	E	*IIR_SimultaneousAlternativeByExpression*		
			IIR_SimultaneousAlternativeByChoices		
			IIR_SimultaneousAlternativeByOthers		
E	*IIR_SimultaneousElsif*				
E	*IIR_WaveformElement*				

8.1 IIR_Tuple

8.1.1 Derived Class Description

The **IIR_Tuple** class represents miscellaneous objects having a predefined set of constituent data elements.

8.1.2 Properties

TABLE 31. IIR_Tuple Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS, Verilog
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR**
Predefined child classes	**IIR_AssociationElement** **IIR_BreakElement** **IIR_CaseStatementAlternative** **IIR_Choice** **IIR_ComponentSpecification** **IIR_ConditionalWaveform** **IIR_ConfigurationItem** **IIR_Designator** **IIR_Elsif** **IIR_EntityClassEntry** **IIR_SelectedWaveform** **IIR_SimultaneousAlternative** **IIR_SimultaneousElsif** **IIR_WaveformElement**
Instantiation?	Indirectly via any of the derived classes of IIR_Tuple
Application-specific data elements	None
Public data elements	None

8.1.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_Tuple.   All of the following methods are atomic.

8.2 IIR_AssociationElement

8.2.1 Derived Class Description

The predefined **IIR_AssociationElement** classes pair a formal and (optional) actual. During elaboration, a list of such association elements serves to associate an actual value with a formal. Association elements are derived into two sub-classes: associations where the actual is an expression (IIR_AssociationElementByExpression) and associations where the actual is open (IIR_AssociationElementOpen). Association elements are organized as individually allocated elements of a list.

8.2.2 Properties

TABLE 32. IIR_AssociationElement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98, VHDL-AMS, Verilog
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Tuple**
Predefined child classes	**IIR_AssociationElementByExpression** **IIR_AssociationElementByOthers** **IIR_AssociationElementOpen**
Instantiation?	Indirectly via either of the classes derived from IIR_AssociationElement
Public data elements	None

8.2.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_AssociationElement. All of the following methods are atomic.

8.2.3.1 Formal Methods

```
void
    set_formal(IIR*        formal);
IIR*
    get_formal();
```

8.3 IIR_AssociationElementByExpression

8.3.1 Derived Class Description

The predefined class **IIR_AssociationElementByExpression** represents either an association between a formal and an explicit actual expression or an association between elements of a composite type and their values within an aggregate.

8.3.2 Properties

TABLE 33. IIR_AssociationElementByExpression Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS, Verilog
IR_Kind enumeration value	**IR_ASSOCIATION_ELEMENT_BY_EXPRESSION**
Parent class	**IIR_AssociationElement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.3.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AssociationElementByExpression object. All of the following methods are atomic.

8.3.3.1 Constructor Method

The constructor initializes an association element by expression object with the undefined source location, undefined formal, undefined actual, and undefined next value.

```
    IIR_AssociationElementByExpression();
```

8.3.3.2 Actual Methods

```
void
    set_actual(    IIR*                    actual);
IIR*
    get_actual();
```

8.3.3.3 Destructor Method

```
~IIR_AssociationElementByExpression();
```

8.4 IIR_AssociationElementByOthers

8.4.1 Derived Class Description

The predefined class **IIR_AssociationElementByOpen** represents an association between all other formals not previously specified and the actual value The actual value is derived from (1) a delayed binding, (2) an initializer associated with the formal interface declaration or (3) the (sub)type of the declaration itself.

8.4.2 Properties

TABLE 34. IIR_AssociationElementOpen Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_ASSOCIATION_ELEMENT_BY_OTHERS**
Parent class	**IIR_AssociationElement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.4.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AssociationElementByOthers object.   All of the following methods are atomic.

8.4.3.1 Constructor Method

The constructor initializes an association element by others object with an undefined source location, an undefined actual, and undefined next value.

```
IIR_AssociationElementByOthers();
```

8.4.3.2 Destructor Method

```
~IIR_AssociationElementByOthers();
```

8.5 IIR_AssociationElementOpen

8.5.1 Derived Class Description

The predefined class **IIR_AssociationElementOpen** represents either an association between a formal and an implicit actual expression or between the elements of a composite type and the value associated with the specified elements within an aggregate. The implicit actual value is derived from (1) a delayed binding, (2) an initializer associated with the formal interface declaration or (3) the (sub)type of the declaration itself.

8.5.2 Properties

TABLE 35. IIR_AssociationElementOpen Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IIR_ASSOCIATION_ELEMENT_OPEN**
Parent class	**IIR_AssociationElement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.5.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AssociationElementOpen object. All of the following methods are atomic.

8.5.3.1 Constructor Method

The constructor initializes an association element by expression object with an undefined source location, an undefined formal, an undefined actual, and undefined next value.

```
IIR_AssociationElementOpen();
```

8.5.3.2 Destructor Method

```
~IIR_AssociationElementOpen();
```

8.6 IIR_BreakElement

8.6.1 Derived Class Description

The predefined **IIR_BreakElement** denotes a single choice within an IIR_BreakElementList.

8.6.2 Properties

TABLE 36. IIR_BreakElement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_BREAK_ELEMENT**
Parent class	**IIR_Tuple**
Predefined child classes	None
Instantiation?	Dynamically
Application-specific data elements	Via extension class

8.6.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_BreakElement object. All of the following methods are atomic.

8.6.3.1 Constructor Method

The constructor initializes choice object with the unspecified source location, and undefiend quantity name and an undefined quantity expression:

```
IIR_BreakElement();
```

8.6.3.2 Quantity Selector Methods

```
void
    set_quantity_selector  IIR*           quantity_selector);
IIR*
    get_quantity_selector();
```

8.6.3.3 Quantity Name Methods

```
void
    set_quantity_name(    IIR*          value);
IIR*
    get_quantity_name();
```

8.6.3.4 Expression Methods

```
void
    set_expression(       IIR*          value);
IIR*
    get_expression();
```

8.6.3.5 Destructor Method

```
    ~IIR_BreakElement();
```

8.7 IIR_CaseStatementAlternative

8.7.1 Derived Class Description

The predefined **IIR_CaseStatementAlternative** represents a choice and implication within a case statement. The choice may explicitly denote single elements of a composite type, lists of elements or may refer to elements which have not already been referenced previously in a case statement alternative list. Objects of case statement alternative class are actually constructed from one of three derived classes corresponding to solitary choices, choice lists and others choices.

8.7.2 Properties

TABLE 37. IIR_CaseStatementAlternative Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS, Verilog
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Tuple**
Predefined child classes	**IIR_CaseStatementAlternativeByExpression** **IIR_CaseStatementAlternativeByChoices** **IIR_CaseStatementAlternativeByOthers**
Instantiation?	Indirectly via any of the derived classes of IIR_CaseStatementAlternative
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_CaseStatementAlternative class

8.7.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_CaseStatementAlternative. All of the following methods are atomic.

8.7.3.1 Predefined Public Data Elements

```
IIR_SequentialStatementList      sequence_of_statements;
```

8.8 IIR_CaseStatementAlternativeByExpression

8.8.1 Derived Class Description

The predefined **IIR_CaseStatementAlternativeByExpressio**n represents a case statement alternative in which the choice is a simple_expression, discrete range (range type), or element simple name (the choice).

8.8.2 Properties

TABLE 38. IIR_CaseStatementAlternativeByExpression Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS, Verilog
IR_Kind enumeration value	**IR_CASE_STATEMENT_ALTERNATIVE_BY_EXPRESSION**
Parent class	**IIR_CaseStatementAlternative**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.8.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_CaseStatementAlternativeByExpression object. All of the following methods are atomic.

8.8.3.1 Constructor Method

The constructor initializes a case statement alternative object with the undefined source location and choice:

```
IIR_CaseStatementAlternativeByExpression();
```

8.8.3.2 Choice Methods

```
void
    set_choice(    IIR*           choice);
IIR*
    get_choice();
```

8.8.3.3 Destructor Method

```
~IIR_CaseStatementAlternativeByExpression();
```

8.9 IIR_CaseStatementAlternativeByChoices

8.9.1 Derived Class Description

The predefined **IIR_CaseStatementAlternativeByChoices** represents a case statement alternative by which two or more choices are simple_expression, discrete range (range type), or element simple name (the choice).

8.9.2 Properties

TABLE 39. **IIR_CaseStatementAlternativeByChoices Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98, VHDL-AMS, Verilog
IR_Kind enumeration value	**IR_CASE_STATEMENT_ALTERNATIVE_BY_CHOICES**
Parent class	**IIR_CaseStatementAlternative**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

8.9.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_CaseStatementAlternativeByChoices object. All of the following methods are atomic.

8.9.3.1 Constructor Method

The constructor initializes a case statement alternative object with the undefined source location and choice:

```
IIR_CaseStatementAlternativeByChoices();
```

8.9.3.2 Destructor Method

```
~IIR_CaseStatementAlternativeByChoices();
```

8.9.4 Predefined Public Data Elements

```
IIR_ChoiceList                    choices;
```

8.10 IIR_CaseStatementAlternativeByOthers

8.10.1 Derived Class Description

The predefined **IIR_CaseStatementAlternativeByExpression** represents a case statement alternative in which the choice implicitly denotes other elements of the case's composite subtype not previously explicit within an IIR_CaseStatementAlternativeList.

8.10.2 Properties

TABLE 40. IIR_CaseStatementAlternativeByOthers Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_CASE_STATEMENT_ALTERNATIVE_BY_OTHERS**
Parent class	**IIR_CaseStatementAlternative**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.10.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_CaseStatementAlternativeByOthers object. All of the following methods are atomic.

8.10.3.1 Constructor Method

The constructor initializes a sequential statement object:

```
IIR_CaseStatementAlternativeByOthers();
```

8.10.3.2 Destructor Method

```
~IIR_CaseStatementAlternativeByOthers();
```

8.11 IIR_Choice

8.11.1 Derived Class Description

The predefined **IIR_Choice** denotes a single choice within a list of two or more alternatives.

8.11.2 Properties

TABLE 41. IIR_Choice Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_CHOICE**
Parent class	**IIR_Tuple**
Predefined child classes	None
Instantiation?	Dynamically
Application-specific data elements	Via extension class

8.11.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_Choice object. All of the following methods are atomic.

8.11.3.1 Constructor Method

The constructor initializes choice object with the unspecified source location and no choice value:

```
    IIR_Choice(    );
```

8.11.3.2 Value Methods

The value may be a simple expression, discrete range or *element*_simple_name.

```
void
    set_value(            IIR*          value);
IIR*
    get_value();
```

8.11.3.3 Destructor Method

```
    ~IIR_Choice();
```

8.12 IIR_ConditionalWaveform

8.12.1 Derived Class Description

The predefined **IIR_ConditionalWaveform** class represents a single conditional waveform element within an IIR_ConditionalWaveformList.

8.12.2 Properties

TABLE 42. IIR_ConditionalWaveform Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_CONDITIONAL_WAVEFORM**
Parent class	**IIR_Tuple**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.12.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConditionalWaveform object. All of the following methods are atomic.

8.12.3.1 Constructor Method

The constructor initializes a conditional waveform object with an undefined condition, undefined next and no waveform elements.

```
IIR_ConditionalWaveform();
```

8.12.3.2 Condition Methods

```
void
    set_condition(        IIR*           condition);
IIR*
    get_condition();
```

8.12.3.3 Destructor Method

```
~IIR_ConditionalWaveform();
```

8.12.4 Predefined Public Data Elements

```
IIR_WaveformList        waveform
```

8.13 IIR_ConfigurationItem

8.13.1 Derived Class Description

The predefined **IIR_ConfigurationItem** class represents a block configuration or component configuration item within an IIR_ConfigurationItemList.

8.13.2 Properties

TABLE 43. IIR_ConfigurationItem Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Tuple**
Predefined child classes	**IIR_BlockConfiguration** **IIR_ComponentConfiguration**
Instantiation?	Indirectly via any of the derived classes of IIR_ConfigurationItem
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_ConfigurationItem class
Public data elements	None

8.13.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_ConfigurationItem. All of the following methods are atomic.

8.14 IIR_BlockConfiguration

8.14.1 Derived Class Description

The predefined **IIR_BlockConfiguration** configures a specific concurrent block (indirectly) within a configuration design unit.

8.14.2 Properties

TABLE 44. IIR_BlockConfiguration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_BLOCK_CONFIGURATIO**N
Parent class	**IIR_ConfigurationItem**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.14.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_BlockConfiguration object. All of the following methods are atomic.

8.14.3.1 Constructor Method

The constructor initializes a block configuration object with the unspecified source location, unspecified block_specification, no use clause items and no configuration items.

```
    IIR_BlockConfiguration();
```

8.14.3.2 Block Specification Methods

```
void
    set_block_specification(      IIR*                    block_specification);
IIR*
    get_block_specification();
```

8.14.3.3 Destructor Method

```
~IIR_BlockConfiguration();
```

8.14.4 Predefined Public Data

```
IIR_DeclarationList           use_clause_list;
IIR_ConfigurationItemList     configuration_item_list;
```

8.15 IIR_ComponentConfiguration

8.15.1 Derived Class Description

The predefined **IIR_ComponentConfiguration** configures a specific component instance (indirectly) within a configuration design unit.

8.15.2 Properties

TABLE 45. IIR_ComponentConfiguration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_COMPONENT_CONFIGURATION**
Parent class	**IIR_ConfigurationItem**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.15.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ComponentConfiguration object. All of the following methods are atomic.

8.15.3.1 Constructor Method

The constructor initializes a component configuration object with no component specification, binding indication or block configuration.

```
IIR_ComponentConfiguration();
```

8.15.3.2 Component Name Method

The component name denotes the component declaration to which all instances in the instantiation list apply.

```
void
    set_component_name(          IIR*          component_name);
IIR*
    get_component_name();
```

8.15.3.3 Entity Aspect Method

The entity aspect refers to the design entity (if any) to be associated with this component. If the entity aspect is NIL, the component configuration is OPEN (unbound). An architecture declaration denotes the associated entity declaration by implication.

```
void
    set_entity_aspect(    IIR_LibraryUnit*      entity_aspect);
IIR_LibraryUnit*
    get_entity_aspect();
```

8.15.3.4 Block Configuration Methods

```
void
    set_block_configuration(IIR_BlockConfiguration*      block_configuration);
IIR_BlockConfiguration*
    get_block_configuration();
```

8.15.3.5 Destructor Method

```
    ~IIR_ComponentConfiguration();
```

8.15.4 Predefined Public Data

Since the domain of all instances is potentially large, the instantiation_list must only include IIR_DesignatorExplicit elements.

```
IIR_DesignatorList              instantiation_list;
IIR_AssociationList             generic_map_aspect;
IIR_AssociationList             port_map_aspect;
```

8.16 IIR_Designator

8.16.1 Derived Class Description

A predefined **IIR_Designator** class names a specific entity with a IIR_DesignatorList.

8.16.2 Properties

TABLE 46. **IIR_Designator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS, Verilog
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Tupler**
Predefined child classes	**IIR_DesignatorExplicit** **IIR_DesignatorByOthers** **IIR_DesignatorByAll**
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.16.3 Predefined Public Methods

All of the following methods must be applied to a valid IIR_Designator object and are atomic.

8.17 IIR_DesignatorExplicit

8.17.1 Derived Class Description

A predefined **IIR_DesignatorExplicit** class names an instance within a IIR_DesignatorList.

8.17.2 Properties

TABLE 47. **IIR_DesignatorExplicit Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS, Verilog
IR_Kind enumeration value	**IR_DESIGNATOR_EXPLICIT**
Parent class	**IIR_Designator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.17.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_DesignatorExplicit object. All of the following methods are atomic.

8.17.3.1 Constructor Method

The constructor initializes an instantiation element with unspecified name.

```
IIR_DesignatorBxplicit();
```

8.17.3.2 Name Methods

```
void
    set_name(      IIR*                          name);
IIR*
    get_name();
```

8.17.3.3 Signature Methods

```
void
    set_signature( IIR_Signature*          signature);
IIR_Signature*
    get_signature();
```

8.17.3.4 Destructor Method

```
    ~IIR_DesignatorExplicit();
```

8.18 IIR_DesignatorByOthers

8.18.1 Derived Class Description

A predefined **IIR_DesignatorByOthers** class names all other instances within a IIR_DesignatorList.

8.18.2 Properties

TABLE 48. IIR_DesignatorByOthers Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS, Verilog
IR_Kind enumeration value	**IR_DESIGNATOR_BY_OTHERS**
Parent class	**IIR_Designator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.18.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_DesignatorByOthers object. All of the following methods are atomic.

8.18.3.1 Constructor Method

The constructor initializes an instantiation element.

```
IIR_DesignatorByOthers();
```

8.18.3.2 Destructor Method

```
~IIR_DesignatorByOthers();
```

8.19 IIR_DesignatorByAll

8.19.1 Derived Class Description

A predefined **IIR_DesignatorByAll** class names all instances within a IIR_DesignatorList.

8.19.2 Properties

TABLE 49. **IIR_DesignatorByAll Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_DESIGNATOR_BY_ALL**
Parent class	**IIR_Designator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.19.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_DesignatorByAll object. All of the following methods are atomic.

8.19.3.1 Constructor Method

The constructor initializes an instantiation element.

```
IIR_DesignatorByAll();
```

8.19.3.2 Destructor Method

```
~IIR_DesignatorByAll();
```

8.20 IIR_Elsif

8.20.1 Derived Class Description

A predefined **IIR_Elsif** class represents one step within a recursive if-then-else statement.

8.20.2 Properties

TABLE 50. IIR_DesignatorByAll Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_Elsif**
Parent class	**IIR_Tuple**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

8.20.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_Elsif object. All of the following methods are atomic.

8.20.3.1 Constructor Method

```
    IIR_Elsif);
```

8.20.3.2 Condition Methods

```
void
    set_condition(IIR*            condition);
IIR*
    get_condition();
```

8.20.3.3 Else Methods

```
void
    set_else_clause(IIR_Elsif*    condition);
IIR_Elsif*
    get_else_clause();
```

8.20.3.4 Destructor Method

```
~IIR_Elsif();
```

8.20.4 Predefined Public Data Elements

```
IIR_SequentialStatementList       then_sequence_of_statements;
```

8.21 IIR_EntityClassEntry

8.21.1 Derived Class Description

A predefined **IIR_EntityClassEntry** represents a specific kind of entity within an IIR_EntityClassList. The IIR_EntityClassList in turn appears only within a group template declaration.

8.21.2 Properties

TABLE 51. IIR_EntityClassEntry Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IIR_ENTITY_CLASS_ENTRY**
Parent class	**IIR_Tuple**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.21.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_EntityClassEntry object. All of the following methods are atomic.

8.21.3.1 Constructor Method

The constructor initializes an entity class entry object with unspecified source location, unspecified entity class and unspecified next entity class entry.

```
IIR_EntityClassEntry();
```

8.21.3.2 Entity Class Methods

```
void
    set_entity_kind(      IR_Kind                        entity_kind);
IR_Kind
    get_entity_kind();
```

8.21.3.3 Box Methods

```
void
    set_boxed(IR_Boolean          is_boxed);
IR_Boolean
    get_boxed();
```

8.21.3.4 Destructor Method

```
    ~IIR_EntityClassEntry();
```

8.22 IIR_SelectedWaveform

8.22.1 Derived Class Description

The predefined **IIR_SelectedWaveform** class represents a selected waveform element within an IIR_SelectedWaveformList. The selected waveform list in turn appears within an IIR_ConcurrentSelectedSignalAssignment.

8.22.2 Properties

TABLE 52. IIR_SelectedWaveform Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SELECTED_WAVEFORM**
Parent class	**IIR_Tuple**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

8.22.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SelectedWaveform object. All of the following methods are atomic.

8.22.3.1 Constructor Method

The constructor initializes a selected waveform object with an unspecified source location, an unspecified choice, a waveform without elements and no next selected waveform element.

```
IIR_SelectedWaveform();
```

8.22.3.2 Value Methods

```
void
    set_choice(          IIR*                    choice);
IIR*
    get_choice();
```

8.22.3.3 Destructor Method

```
~IIR_SelectedWaveform();
```

8.22.4 Predefined Public Data Element

```
IIR_WaveformList          waveform;
```

8.23 IIR_SimultaneousAlternative

8.23.1 Derived Class Description

The predefined **IIR_SimultaneousAlternative** represents a choice and implication within a simultaneous case statement. The choice may explicitly denote single elements of a composite type, lists of elements or may refer to elements which have not already been referenced previously in a case statement alternative list. Objects of case statement alternative class are actually constructed from one of three derived classes corresponding to solitary choices, choice lists and others choices.

8.23.2 Properties

TABLE 53. IIR_CaseStatementAlternative Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Tuple**
Predefined child classes	**IIR_SimultaneousAlternativeByExpression** **IIR_SimultaneousAlternativeByChoices** **IIR_SimultaneousAlternativeByOthers**
Instantiation?	Indirectly via any of the derived classes of IIR_SimultaneousAlternative
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_SimultaneousAlternative class

8.23.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_SimultaneousAlternative. All of the following methods are atomic.

8.23.3.1 Constructor Method

The constructor initializes a selected waveform object with an unspecified source location, an unspecified choice, a waveform without elements and no next selected waveform element.

```
IIR_SimultaneousAlternative();
```

8.23.3.2 Destructor Method

```
~IIR_SimultaneousAlternative();
```

8.23.3.3 Predefined Public Data Elements

```
IIR_SimultaneousStatementList     sequence_of_statements;
```

8.24 IIR_SimultaneousAlternativeByExpression

8.24.1 Derived Class Description

The predefined **IIR_SimultaneousAlternativeByExpressio**n represents a simultaneous alternative in which the choice is a simple_expression, discrete range (range type), or element simple name (the choice).

8.24.2 Properties

TABLE 54. IIR_SimultaneousAlternativeByExpression Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMULTANEOUS_ALTERNATIVE_BY_EXPRESSION**
Parent class	**IIR_SimultaneousAlternative**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.24.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousAlternativeByExpression object. All of the following methods are atomic.

8.24.3.1 Constructor Method

The constructor initializes a simultaneous alternative object with the undefined source location and choice:

```
IIR_SimultaneousAlternativeByExpression();
```

8.24.3.2 Choice Methods

```
void
    set_choice(    IIR*           choice);
IIR*
    get_choice();
```

8.24.3.3 Destructor Method

```
~IIR_SimultaneousAlternativeByExpression();
```

8.25 IIR_SimultaneousAlternativeByChoices

8.25.1 Derived Class Description

The predefined **IIR_SimultaneousAlternativeByChoices** represents a case statement alternative by which two or more choices are simple_expression, discrete range (range type), or element simple name (the choice).

8.25.2 Properties

TABLE 55. **IIR_SimultaneousAlternativeByChoices Properties**

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMULTANEOUS_ALTERNATIVE_BY_CHOICES**
Parent class	**IIR_SimultaneousAlternative**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

8.25.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousAlternativeByChoices object. All of the following methods are atomic.

8.25.3.1 Constructor Method

The constructor initializes a simultaneous alternative object with the undefined source location and choice:

```
IIR_SimultaneousAlternativeByChoices( );
```

8.25.3.2 Destructor Method

```
~IIR_SimultaneousAlternativeByChoices();
```

8.25.4 Predefined Public Data Elements

```
IIR_ChoiceList                    choices;
```

8.26 IIR_SimultaneousAlternativeByOthers

8.26.1 Derived Class Description

The predefined **IIR_SimultaneousAlternativeByOthers** represents a simultaneous alternative in which the choice implicitly denotes other elements of the case's composite subtype not previously explicit within an IIR_SimultaneousAlternativeList.

8.26.2 Properties

TABLE 56. **IIR_SimultaneousAlternativeByOthers Properties**

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMULTANEOUS_ALTERNATIVE_BY_OTHERS**
Parent class	**IIR_SimultaneousAlternative**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.26.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousAlternativeByOthers object. All of the following methods are atomic.

8.26.3.1 Constructor Method

```
IIR_SimultaneousAlternativeByOthers();
```

8.26.3.2 Destructor Method

```
~IIR_SimultaneousAlternativeByOthers();
```

8.27 IIR_SimultaneousElsif

8.27.1 Derived Class Description

A predefined **IIR_SimultaneousElsif** class represents one step within a recursive simultaneous if-then-else statement.

8.27.2 Properties

TABLE 57. IIR_DesignatorByAll Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IIR_SimultaneousElsif**
Parent class	**IIR_Tuple**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

8.27.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousElsif object. All of the following methods are atomic.

8.27.3.1 Constructor Method

The constructor initializes a simultaneous elsif element.

```
IIR_SimultaneousElsif();
```

8.27.3.2 Condition Methods

```
void
    set_condition(IIR*            condition);
IIR*
    get_condition();
```

8.27.3.3 Else Methods

```
void
    set_else_clause(IIR_Elsif*    else_clause);
IIR_Elsif*
    get_else_clause();
```

8.27.3.4 Destructor Method

```
    ~IIR_SimultaneousElsif();
```

8.27.4 Predefined Public Data Elements

```
IIR_SequentialStatementList       then_sequence_of_statements;
```

8.28 IIR_WaveformElement

8.28.1 Derived Class Description

The predefined class **IIR_WaveformElement** represents a value and time tuple within an IIR_WaveformList. Such a waveform list appears directly or indirectly within sequential and concurrent signal assignment statements.

8.28.2 Properties

TABLE 58. IIR_WaveformElement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_WAVEFORM_ELEMENT**
Parent class	**IIR_Tuple**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

8.28.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_WaveformElement object. All of the following methods are atomic.

8.28.3.1 Constructor Method

The constructor initializes a waveform element object with an unspecified source, an unspecified value expression, an unspecified time expression and no next waveform element.

```
IIR_WaveformElement();
```

8.28.3.2 Value Methods

Value methods refer to the value being assigned; a NIL value corresponds to a NULL assignment.

```
void
    set_value(     IIR*                    value);
IIR*
    get_value();
```

8.28.3.3 Time Methods

```
void
    set_time(       IIR*                              time);
IIR*
    get_time();
```

8.28.3.4 Destructor Method

```
    ~IIR_WaveformElement();
```

CHAPTER 9 *IIR_List Derived Classes*

List classes provide predefined methods providing functionality such as identifying the first element in a list, determining the length of a list, and iterating forward or backward through a elements of a list. Applications may add application-specific methods to IIRBase_List and all of the classes descended from it using extension class layers, however no new data items may be added directly to lists.

List classes are generally embedded directly into the public part of other IIR classes. This is in contrast to most other IIR classes, which are independently allocated and referenced by a pointer. Embedding helps to reduce memory consumption and allocation overhead, but *does not allow* addition of data elements within list extension classes.

TABLE 59. Class derivation hierarchy from IIR_List

	Level 3 Derived Classes (only classes derived from IIR_List)		**Level 4 Derived Classes**		**Level 5 Derived Classes**
E	*IIR_AssociationList*				
E	*IIR_AttributeSpecificationList*				
E	*IIR_BreakList*				
E	*IIR_CaseStatementAlternativeList*				
E	*IIR_ChoiceList*				
E	*IIR_CommentList*				
E	*IIR_ConcurrentStatementList*				
E	*IIR_ConditionalWaveformList*				
E	*IIR_ConfigurationItemList*				
E	*IIR_DeclarationList*				
E	*IIR_DesignFileList*				
E	*IIR_DesignatorList*				

TABLE 59. Class derivation hierarchy from IIR_List

E	*IIR_ElementDeclarationList*				
E	*IIR_NatureElementDeclarationList*				
E	*IIR_EntityClassEntryList*				
E	*IIR_EnumerationLiteralList*				
E	*IIR_GenericList*				
E	*IIR_InterfaceList*				
E	*IIR_LibraryUnitList*				
E	*IIR_PortList*				
E	*IIR_SelectedWaveformList*				
E	*IIR_SequentialStatementList*				
E	*IIR_SimultaneousAlternativeList*				
E	*IIR_SimultaneousStatementList*				
E	*IIR_StatementList*				
E	*IIR_UnitList*				
E	*IIR_WaveformList*				

9.1 IIR_List

9.1.1 Derived Class Description

The IIR_Lists class represents a collection of zero or more dynamically allocated elements having a specified class or common parent class.

TABLE 60. IIR_List Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR**
Predefined child classes	**IIR_AssociationList** **IIR_AttributeSpecificationList** **IIR_BreakList** **IIR_Case StatementAlternativeList** **IIR_ChoiceList** **IIR_CommentList** **IIR_ConcurrentStatementList** **IIR_ConditionalWaveformList** **IIR_ConfigurationItemList** **IIR_DeclarationList** **IIR_DesignFileList** **IIR_DesignatorList** **IIR_ElementDeclarationList** **IIR_NatureElementDeclarationList** **IIR_EntityClassEntryList** **IIR_EnumerationLiteralList** **IIR_GenericList** **IIR_InterfaceList** **IIR_LibraryUnitList** **IIR_PortList** **IIR_SelectedWaveformList** **IIR_SequentialStatementList** **IIR_SimultaneousAlternativeList** **IIR_SimultaneousStatementList** **IIR_StatementList** **IIR_UnitList** **IIR_WaveformList**
Instantiation?	Indirectly via any of the derived classes of IIR_List
Application-specific data elements	Not allowed
Public data elements	None

9.1.2 Properties

9.1.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_List. All of the following methods are atomic.

9.1.3.1 List Properties

The length property returns the number of elements present in a list.

```
IIR_Int32
    number_of_elements();
```

9.2 IIR_AssociationList

9.2.1 Derived Class Description

The **IIR_AssociationList** class represents ordered sets containing zero or more IIR_AssociationElements. Association lists are either used at an elaboration interface to associate actuals with formals or to represent elements of an aggregate value.

9.2.2 Properties

TABLE 61. IIR_AssociationList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ASSOCIATION_LIST**
Parent class	IIR_List
Predefined child classes	None
Instantiation?	Indirectly as part of other constructs which include an association list.
Application-specific data elements	Not allowed
Public data elements	None

9.2.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AssociationList object. All of the following methods are atomic.

9.2.3.1 Constructor Method

The default constructor results in a list with no elements.

```
IIR_AssociationList();
```

9.2.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert

method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_AssociationElement*      element);

void
    append_element(             IIR_AssociationElement*      element);
IR_Boolean
    insert_after_element(       IIR_AssociationElement*      existing_element,
                                IIR_AssociationElement*      new_element);
IR_Boolean
    insert_before_element(      IIR_AssociationElement*      existing_element,
                                IIR_AssociationElement*      new_element);
IR_Boolean
    remove_element(             IIR_AssociationElement*      existing_element);
IIR_AssociationElement*
    get_successor_element(      IIR_AssociationElement*      existing_element);
IIR_AssociationElement*
    get_predecessor_element(    IIR_AssociationElement*      element);
IIR_AssociationElement*
    get_first_element();
IIR_AssociationElement*
    get_nth_element(            IR_Int32                     index);
IIR_AssociationElement*
    get_last_element();
IR_Int32
    get_element_position(       IIR_AssociationElement*      element);
```

9.2.3.3 List Destructor Method

The list destructor method deletes all elements of the association list, then deletes the list object itself.

```
void
    ~IIR_AssociationList();
```

9.3 IIR_AttributeSpecificationList

9.3.1 Derived Class Description

The **IIR_AttributeSpecificationList** class represents ordered sets containing zero or more IIR_AttributeSpecifications.

9.3.2 Properties

TABLE 62. IIR_AttributeSpecificationList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ATTRIBUTE_SPECIFICATION_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly as part of other constructs which include an association list.
Application-specific data elements	Not allowed
Public data elements	None

9.3.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AttributeSpecificationList object. All of the following methods are atomic.

9.3.3.1 Constructor Method

The default constructor results in a list with no elements.

```
IIR_AttributeSpecificationList();
```

9.3.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method

returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(              IIR_AttributeSpecification*   element);
void
    append_element(               IIR_AttributeSpecification*   element);
IR_Boolean
    insert_after_element(         IIR_AttributeSpecification*   existing_element,
                                  IIR_AttributeSpecification*   new_element);
IR_Boolean
    insert_before_element(        IIR_AttributeSpecification*   existing_element,
                                  IIR_AttributeSpecification*   new_element);
IR_Boolean
    remove_element(               IIR_AttributeSpecification*   existing_element);
IIR_AttributeSpecification*
    get_successor_element(        IIR_AttributeSpecification*   existing_element);
IIR_AttributeSpecification*
    get_predecessor_element(      IIR_AttributeSpecification*   element);
IIR_AttributeSpecification*
    get_first_element();
IIR_AttributeSpecification*
    get_nth_element(              IR_Int32                      index);
IIR_AttributeSpecification*
    get_last_element();
IR_Int32
    get_element_position(         IIR_AttributeSpecification*   element);
```

9.3.3.3 List Destructor Method

The list destructor method deletes all elements of the attribute specification list, then deletes the list object itself.

```
void
    ~IIR_AttributeSpecificationList();
```

9.4 IIR_BreakList

9.4.1 Derived Class Description

The **IIR_BreakList** class represents ordered sets containing zero or more **IIR_BreakElement**s.

9.4.2 Properties

TABLE 63. IIR_BreakList Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_BREAK_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly as part of other constructs which include an association list.
Application-specific data elements	Not allowed
Public data elements	None

9.4.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_BreakList object. All of the following methods are atomic.

9.4.3.1 Constructor Method

The default constructor results in a list with no elements.

```
IIR_BreakList();
```

9.4.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***,

get_nth_element() and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(          IIR_BreakElement*         element);
void
    append_element(           IIR_BreakElement*         element);
IR_Boolean
    insert_after_element(     IIR_BreakElement*         existing_element,
                              IIR_BreakElement*         new_element);
IR_Boolean
    insert_before_element(    IIR_BreakElement*         existing_element,
                              IIR_BreakElement*         new_element);
IR_Boolean
    remove_element(           IIR_BreakElement*         existing_element);
IIR_BreakElement*
    get_successor_element(    IIR_BreakElement*         existing_element);
IIR_BreakElement*
    get_predecessor_element(  IIR_BreakElement*         element);
IIR_BreakElement*
    get_first_element();
IIR_BreakElement*
    get_nth_element(          IR_Int32                  index);
IIR_BreakElement*
    get_last_element();
IR_Int32
    get_element_position(     IIR_BreakElement*         element);
```

9.4.3.3 List Destructor Method

The list destructor method deletes all elements of the break list, then deletes the list object itself.

```
void
    ~IIR_BreakList();
```

9.5 IIR_CaseStatementAlternativeList

9.5.1 Derived Class Description

The predefined **IIR_CaseStatementAlternativeList** class represents ordered sets containing zero or more IIR_CaseStatementAlternative objects. Case statement alternative lists are used within case statements to denote the list of choice, implication pairs.

9.5.2 Properties

TABLE 64. IIR_CaseStatementAlternativeList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CASE_STATEMENT_ALTERNATIVE_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly as a public data element within the IIR_CaseStatement class
Application-specific data elements	Not allowed
Public data elements	None

9.5.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_CaseStatementAlternativeList object. All of the following methods are atomic.

9.5.3.1 Constructor Method

The default constructor results in a list with no elements.

```
IIR_CaseStatementAlternativeList();
```

9.5.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert

method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(          IIR_CaseStatementAlternative*       element);
void
    append_element(           IIR_CaseStatementAlternative*       element);
IR_Boolean
    insert_after_element(     IIR_CaseStatementAlternative*       existing_element,
                              IIR_CaseStatementAlternative*       new_element);
IR_Boolean
    insert_before_element(    IIR_CaseStatementAlternative*       existing_element,
                              IIR_CaseStatementAlternative*       new_element);
IR_Boolean
    remove_element(           IIR_CaseStatementAlternative*       existing_element);
IIR_CaseStatementAlternative*
    get_successor_element(    IIR_CaseStatementAlternative*       existing_element);
IIR_CaseStatementAlternative*
    get_predecessor_element(  IIR_CaseStatementAlternative*       element);
IIR_CaseStatementAlternative*
    get_first_element();
IIR_CaseStatementAlternative*
    get_nth_element(          IR_Int32                            index);
IIR_CaseStatementAlternative*
    get_last_element();
IR_Int32
    get_element_position(     IIR_CaseStatementAlternative*       element);
```

9.5.3.3 List Destructor Method

The list destructor method deletes all elements of the alternative list, then deletes the list object itself.

```
void
    ~IIR_CaseStatementAlternativeList();
```

9.6 IIR_ChoiceList

9.6.1 Derived Class Description

The predefined **IIR_ChoiceList** class represents ordered sets containing zero or more IIR_Choice objects. Choice lists are used within case statements to denote lists containing two or more choices.

9.6.2 Properties

TABLE 65. **IIR_ChoiceList Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CHOICE_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly as a public data element within the IIR_CaseStatementAlternative class
Application-specific data elements	Not allowed
Public data elements	None

9.6.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ChoiceList object. All of the following methods are atomic.

9.6.3.1 Constructor Method

The default constructor results in a list with no elements.

```
IIR_ChoiceList();
```

9.6.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method

returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_Choice*              element);
void
    append_element(             IIR_Choice*              element);
IR_Boolean
    insert_after_element(       IIR_Choice*              existing_element,
                                IIR_Choice*              new_element);
IR_Boolean
    insert_before_element(      IIR_Choice*              existing_element,
                                IIR_Choice*              new_element);
IR_Boolean
    remove_element(             IIR_Choice*              existing_element);
IIR_Choice*
    get_successor_element(      IIR_Choice*              existing_element);
IIR_Choice*
    get_predecessor_element(    IIR_Choice*              element);
IIR_Choice*
    get_first_element();
IIR_Choice*
    get_nth_element(            IR_Int32                 index);
IIR_Choice*
    get_last_element();
IIR_Int32
    get_element_position(       IIR_Choice*              element);
```

9.6.3.3 List Destructor Method

The list destructor method deletes all elements of the choice list, then deletes the list object itself.

```
void
    ~IIR_ChoiceList();
```

9.7 IIR_CommentList

9.7.1 Derived Class Description

The predefined **IIR_CommentList** represent ordered sets containing zero or more IIR_Comments. Such comment lists appear only within IIR_DesignFiles within the predefined IIR class hierarchy.

9.7.2 Properties

TABLE 66. IIR_CommentList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_COMMENT_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly as part of IIR_DesignFiles
Application-specific data elements	Not allowed
Public data elements	None

9.7.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_CommentList object. All of the following methods are atomic.

9.7.3.1 Constructor Method

The default constructor results in a comment list with no comment elements.

```
IIR_CommentList();
```

9.7.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method

returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_Comment*          element);

void
    append_element(             IIR_Comment*          element);
IR_Boolean
    insert_after_element(       IIR_Comment*          existing_element,
                                IIR_Comment*          new_element);
IR_Boolean
    insert_before_element(      IIR_Comment*          existing_element,
                                IIR_Comment*          new_element);
IR_Boolean
    remove_element(             IIR_Comment*          existing_element);
IIR_Comment*
    get_successor_element(      IIR_Comment*          existing_element);
IIR_Comment*
    get_predecessor_element(    IIR_Comment*          element);
IIR_Comment*
    get_first_element();
IIR_Comment*
    get_nth_element(            IR_Int32              index);
IIR_Comment*
    get_last_element();
IIR_Int32
    get_element_position(       IIR_Comment*          element);
```

9.7.3.3 List Destructor Method

The list destructor method deletes all elements of the association list, then deletes the list object itself.

```
void
    ~IIR_CommentList();
```

9.8 IIR_ConcurrentStatementList

9.8.1 Derived Class Description

The predefined **IIR_ConcurrentStatementList** class represents ordered sets containing zero or more IIR_ConcurrentStatements. Such lists are found directly within entities, architectures block statements and generate statements.

9.8.2 Properties

TABLE 67. IIR_ConcurrentStatementList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONCURRENT_STATEMENT_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects are an element of other IIR classes
Application-specific data elements	Not allowed
Public data elements	None

9.8.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConcurrentStatementList object. All of the following methods are atomic.

9.8.3.1 Constructor Method

The default constructor results in a concurrent statement list with no concurrent statement elements.

```
IIR_ConcurrentStatementList();
```

9.8.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method

returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_ConcurrentStatement*      element);
void
    append_element(             IIR_ConcurrentStatement*      element);
IR_Boolean
    insert_after_element(       IIR_ConcurrentStatement*      existing_element,
                                IIR_ConcurrentStatement*      new_element);
IR_Boolean
    insert_before_element(      IIR_ConcurrentStatement*      existing_element,
                                IIR_ConcurrentStatement*      new_element);
IR_Boolean
    remove_element(             IIR_ConcurrentStatement*      existing_element);
IIR_ConcurrentStatement*
    get_successor_element(      IIR_ConcurrentStatement*      existing_element);
IIR_ConcurrentStatement*
    get_predecessor_element(    IIR_ConcurrentStatement*      element);
IIR_ConcurrentStatement*
    get_first_element();
IIR_ConcurrentStatement*
    get_nth_element(            IR_Int32                      index);
IIR_ConcurrentStatement*
    get_last_element();
IR_Int32
    get_element_position(       IIR_ConcurrentStatement*      element);
```

9.8.3.3 List Destructor Method

The list destructor method deletes all concurrent statements within the concurrent statement list, then deletes the concurrent statement list object itself.

```
void
    ~IIR_ConcurrentStatementList();
```

9.9 IIR_ConditionalWaveformList

9.9.1 Derived Class Description

The predefined **IIR_ConditionalWaveformList** represents ordered sets containing zero or more IIR_ConditionalWaveform elements. Such lists appear directly within IIR_ConcurrentConditionalSignalAssignment statements.

9.9.2 Properties

TABLE 68. IIR_ConditionalWaveformList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONDITIONAL_WAVEFORM_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects are part of concurrent conditional signal assignment statements
Application-specific data elements	Not allowed
Public data elements	None

9.9.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConditionalWaveformList object. All of the following methods are atomic.

9.9.3.1 Constructor Method

The default constructor results in a conditional waveform list with no conditional waveform elements.

```
IIR_ConditionalWaveformList();
```

9.9.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert

method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(             IIR_ConditionalWaveform*      element);

void
    append_element(              IIR_ConditionalWaveform*      element);
IR_Boolean
    insert_after_element(        IIR_ConditionalWaveform*      existing_element,
                                 IIR_ConditionalWaveform*      new_element);
IR_Boolean
    insert_before_element(       IIR_ConditionalWaveform*      existing_element,
                                 IIR_ConditionalWaveform*      new_element);
IR_Boolean
    remove_element(              IIR_ConditionalWaveform*      existing_element);
IIR_ConditionalWaveform*
    get_successor_element(       IIR_ConditionalWaveform*      existing_element);
IIR_ConditionalWaveform*
    get_predecessor_element(     IIR_ConditionalWaveform*      element);
IIR_ConditionalWaveform*
    get_first_element();

IIR_ConditionalWaveform*
    get_nth_element(             IR_Int32                      index);
IIR_ConditionalWaveform*
    get_last_element();
IR_Int32
    get_element_position(        IIR_ConditionalWaveform*      element);
```

9.9.3.3 List Destructor Method

The list destructor method deletes all conditional waveform elements of the within the conditional waveform list, then deletes the condiotional waveform list object itself.

```
void
    ~IIR_ConditionalWaveformList();
```

9.10 IIR_ConfigurationItemList

9.10.1 Derived Class Description

The predefined **IIR_ConfigurationItemList** class represents ordered sets containing zero or more IIR_ConfigurationItems (block or component configurations). Configuration item lists appear directly within block configurations.

9.10.2 Properties

TABLE 69. IIR_ConfigurationItemList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONFIGURATION_ITEM_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, IIR_ConfigurationItemLists are predefined public data elements within block configurations
Application-specific data elements	Not allowed
Public data elements	None

9.10.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConfigurationItemList object. All of the following methods are atomic.

9.10.3.1 Constructor Method

The default constructor results in a configuration item list with no configuration items.

```
IIR_ConfigurationItemList();
```

9.10.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert

method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_ConfigurationItem*        element);
void
    append_element(             IIR_ConfigurationItem*        element);
IR_Boolean
    insert_after_element(       IIR_ConfigurationItem*        existing_element,
                                IIR_ConfigurationItem*        new_element);
IR_Boolean
    insert_before_element(      IIR_ConfigurationItem*        existing_element,
                                IIR_ConfigurationItem*        new_element);
IR_Boolean
    remove_element(             IIR_ConfigurationItem*        existing_element);
IIR_ConfigurationItem*
    get_successor_element(      IIR_ConfigurationItem*        existing_element);
IIR_ConfigurationItem*
    get_predecessor_element(    IIR_ConfigurationItem*        element);
IIR_ConfigurationItem*
    get_first_element();
IIR_ConfigurationItem*
    get_nth_element(            IR_Int32                      index);
IIR_ConfigurationItem*
    get_last_element();
IR_Int32
    get_element_position(       IIR_ConfigurationItem*        element);
```

9.10.3.3 List Destructor Method

The list destructor method deletes all configuration items within the configuration item list, then deletes the list object itself.

```
void
    ~IIR_ConfigurationItemList();
```

9.11 IIR_DeclarationList

9.11.1 Derived Class Description

The predefined **IIR_DeclarationList** class represents ordered sets containing zero or more IIR_Declarations (such declarations broadly include declarations, specifications and use clauses). Such declaration lists are directly incorporated into many other predefined IIR classes.

9.11.2 Properties

TABLE 70. IIR_DeclarationList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_DECLARATION_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_DeclarationList class are pre-defined public data elements of many other IIR classes
Application-specific data elements	Not allowed
Public data elements	None

9.11.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_DeclarationList object. All of the following methods are atomic.

9.11.3.1 Constructor Method

The default constructor results in a declaration list with no declarations.

```
IIR_DeclarationList();
```

9.11.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert

method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_Declaration*      element);
void
    append_element(             IIR_Declaration*      element);
IR_Boolean
    insert_after_element(       IIR_Declaration*      existing_element,
                                IIR_Declaration*      new_element);
IR_Boolean
    insert_before_element(      IIR_Declaration*      existing_element,
                                IIR_Declaration*      new_element);
IR_Boolean
    remove_element(             IIR_Declaration*      existing_element);
IIR_Declaration*
    get_successor_element(      IIR_Declaration*      existing_element);
IIR_Declaration*
    get_predecessor_element(    IIR_Declaration*      element);
IIR_Declaration*
    get_first_element();
IIR_Declaration*
    get_nth_element(            IR_Int32              index);
IIR_Declaration*
    get_last_element();
IR_Int32
    get_element_position(       IIR_Declaration*      element);
```

9.11.3.3 List Destructor Method

The list destructor method deletes all elements of the declaration list, then deletes the list object itself.

```
void
~IIR_DeclarationList();
```

9.12 IIR_DesignFileList

9.12.1 Derived Class Description

The predefined IIR_DesignFileList class represents ordered sets containing zero or more IIR_DesignFiles. Within the predefined IIR data structures, IIR_DesignFileLists only serve as a global, static data element from which all files within the IIR data structures may eventually be reached.

9.12.2 Properties

TABLE 71. IIR_DesignFileList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_DESIGN_FILE_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Statically
Application-specific data elements	Not allowed
Public data elements	None

9.12.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_DesignFileList object. All of the following methods are atomic.

9.12.3.1 Constructor Method

The default constructor results in a design file list with no design files.

```
IIR_DesignFileList();
```

9.12.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method

returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_DesignFile*       element);

void
    append_element(             IIR_DesignFile*       element);
IR_Boolean
    insert_after_element(       IIR_DesignFile*       existing_element,
                                IIR_DesignFile*       new_element);
IR_Boolean
    insert_before_element(      IIR_DesignFile*       existing_element,
                                IIR_DesignFile*       new_element);
IR_Boolean
    remove_element(             IIR_DesignFile*       existing_element);
IIR_DesignFile*
    get_successor_element(      IIR_DesignFile*       existing_element);
IIR_DesignFile*
    get_predecessor_element(    IIR_DesignFile*       element);
IIR_DesignFile*
    get_first_element();
IIR_DesignFile*
    get_nth_element(            IR_Int32              index);
IIR_DesignFile*
    get_last_element();
IR_Int32
    get_element_position(       IIR_DesignFile*       element);
```

9.12.3.3 List Destructor Method

The list destructor method deletes all design files within the design file list, then deletes the list object itself. *Caution: when applied to the statically allocated list of all design files, this destructor deletes an entire IIR database*.

```
void
    ~IIR_DesignFileList();
```

9.13 IIR_DesignatorList

9.13.1 Derived Class Description

The predefined **IIR_DesignatorList** class represents an ordered sets containing zero or more IIR_Designator tuples.

9.13.2 Properties

TABLE 72. IIR_DesignatorList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_DESIGNATOR_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	?
Application-specific data elements	Not allowed
Public data elements	None

9.13.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_DesignatorList object. All of the following methods are atomic.

9.13.3.1 Constructor Method

The default constructor results in a designator list with no designators.

```
IIR_DesignatorList();
```

9.13.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method

returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_Designator*       element);

void
    append_element(             IIR_Designator*       element);
IR_Boolean
    insert_after_element(       IIR_Designator*       existing_element,
                                IIR_Designator*       new_element);
IR_Boolean
    insert_before_element(      IIR_Designator*       existing_element,
                                IIR_Designator*       new_element);
IR_Boolean
    remove_element(             IIR_Designator*       existing_element);
IIR_Designator*
    get_successor_element(      IIR_Designator*       existing_element);
IIR_Designator*
    get_predecessor_element(    IIR_Designator*       element);
IIR_Designator*
    get_first_element();
IIR_Designator*
    get_nth_element(            IR_Int32              index);
IIR_Designator*
    get_last_element();
IR_Int32
    get_element_position(       IIR_Designator*       element);
```

9.13.3.3 List Destructor Method

The list destructor method deletes all designators in the designator list, then deletes the list object itself.

```
void
    ~IIR_DesignatorList();
```

9.14 IIR_ElementDeclarationList

9.14.1 Derived Class Description

The predefined **IIR_ElementDeclarationList** class represents ordered sets containing zero or more IIR_ElementDeclarations. Element declaration lists appear as public data elements within record type definitions.

9.14.2 Properties

TABLE 73. IIR_ElementDeclarationList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ELEMENT_DECLARATION_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_ElementDeclarationList class are predefined public data elements within an IIR_RecordTypeDefinition
Application-specific data elements	Not allowed
Public data elements	None

9.14.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ElementDeclarationList object. All of the following methods are atomic.

9.14.3.1 Constructor Method

The default constructor results in a element declaration list with no element declarations.

```
IIR_ElementDeclarationList();
```

9.14.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method

does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(              IIR_ElementDeclaration*       element);
void
    append_element(               IIR_ElementDeclaration*       element);
IR_Boolean
    insert_after_element(         IIR_ElementDeclaration*       existing_element,
                                  IIR_ElementDeclaration*       new_element);
IR_Boolean
    insert_before_element(        IIR_ElementDeclaration*       existing_element,
                                  IIR_ElementDeclaration*       new_element);
IR_Boolean
    remove_element(               IIR_ElementDeclaration*       existing_element);
IIR_ElementDeclaration*
    get_successor_element(        IIR_ElementDeclaration*       existing_element);
IIR_ElementDeclaration*
    get_predecessor_element(      IIR_ElementDeclaration*       element);
IIR_ElementDeclaration*
    get_first_element();
IIR_ElementDeclaration*
    get_nth_element(              IR_Int32                      index);
IIR_ElementDeclaration*
    get_last_element();
IR_Int32
    get_element_position(         IIR_ElementDeclaration*       element);
```

9.14.3.3 List Destructor Method

The list destructor method deletes all elements of the element declaration list, then deletes the list object itself.

```
void
    ~IIR_ElementDeclarationList();
```

9.15 IIR_NatureElementDeclarationList

9.15.1 Derived Class Description

The predefined **IIR_NatureElementDeclarationList** class represents ordered sets containing zero or more IIR_ElementDeclarations. Element declaration lists appear as public data elements within record type definitions.

9.15.2 Properties

TABLE 74. IIR_NatureElementDeclarationList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_NATURE_ELEMENT_DECLARATION_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_NatureElementDeclarationList class are predefined public data elements within an IIR_NatureRecordTypeDefinition
Application-specific data elements	Not allowed
Public data elements	None

9.15.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_NatureElementDeclarationList object. All of the following methods are atomic.

9.15.3.1 Constructor Method

The default constructor results in a element declaration list with no element declarations.

```
IIR_EnaturelementDeclarationList();
```

9.15.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The

remove_element() method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(             IIR_NatureElementDeclaration* element);
void
    append_element(              IIR_NatureElementDeclaration* element);
IR_Boolean
    insert_after_element(        IIR_NatureElementDeclaration* existing_element,
                                 IIR_NatureElementDeclaration* new_element);
IR_Boolean
    insert_before_element(       IIR_NatureElementDeclaration* existing_element,
                                 IIR_NatureElementDeclaration* new_element);
IR_Boolean
    remove_element(              IIR_NatureElementDeclaration* existing_element);
IIR_NatureElementDeclaration*
    get_successor_element(       IIR_NatureElementDeclaration* existing_element);
IIR_NatureElementDeclaration*
    get_predecessor_element(     IIR_NatureElementDeclaration* element);
IIR_NatureElementDeclaration*
    get_first_element();
IIR_NatureElementDeclaration*
    get_nth_element(             IR_Int32                      index);
IIR_NatureElementDeclaration*
    get_last_element();
IR_Int32
    get_element_position(        IIR_NatureElementDeclaration* element);
```

9.15.3.3 List Destructor Method

The list destructor method deletes all elements of the element declaration list, then deletes the list object itself.

```
void
    ~IIR_NatureElementDeclarationList();
```

9.16 IIR_EntityClassEntryList

9.16.1 Derived Class Description

The predefined **IIR_EntityClassEntryList** represents ordered sets containing zero or more IIR_EntityClassEntry objects. Entity class entry lists appear as predefined public data elements within IIR_GroupTemplateDeclarations.

9.16.2 Properties

TABLE 75. IIR_EntityClassEntryList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ENTITY_CLASS_ENTRY_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_EntityClassEntryList are predefined public data elements within IIR_GroupTemplateDeclarations
Application-specific data elements	Not allowed
Public data elements	None

9.16.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_EntityClassEntryList object. All of the following methods are atomic.

9.16.3.1 Constructor Method

The default constructor results in an entity class entry list with no entity class entry elements.

```
IIR_EntityClassEntryList();
```

9.16.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method

does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_EntityClassEntry*      element);
void
    append_element(             IIR_EntityClassEntry*      element);
IR_Boolean
    insert_after_element(       IIR_EntityClassEntry*      existing_element,
                                IIR_EntityClassEntry*      new_element);
IR_Boolean
    insert_before_element(      IIR_EntityClassEntry*      existing_element,
                                IIR_EntityClassEntry*      new_element);
IR_Boolean
    remove_element(             IIR_EntityClassEntry*      existing_element);
IIR_EntityClassEntry*
    get_successor_element(      IIR_EntityClassEntry*      existing_element);
IIR_EntityClassEntry*
    get_predecessor_element(    IIR_EntityClassEntry*      element);
IIR_EntityClassEntry*
    get_first_element();
IIR_EntityClassEntry*
    get_nth_element(            IR_Int32                   index);
IIR_EntityClassEntry*
    get_last_element();
IR_Int32
    get_element_position(       IIR_EntityClassEntry*      element);
```

9.16.3.3 List Destructor Method

The list destructor method deletes all entity class entries within the entity class entry list, then deletes the list object itself.

```
void
    ~IIR_EntityClassEntryList();
```

9.17 IIR_EnumerationLiteralList

9.17.1 Derived Class Description

The predefined **IIR_EnumerationLiteralList** class represents ordered sets containing zero or more IIR_EnumerationLiterals. Enumerational literal lists are found as predefined public data elements within IIR_EnumerationTypeDefinition and IIR_EnumerationSubtypeDefinition classes.

9.17.2 Properties

TABLE 76. IIR_EnumerationLiteralList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ENUMERATION_LITERAL_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_EnumerationLiteralList class are predefined public data elements of IIR_EnumerationTypeDefinition and IIR_EnumerationSubtypeDefinition classes
Application-specific data elements	Not allowed
Public data elements	None

9.17.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_EnumerationLiteralList object. All of the following methods are atomic.

9.17.3.1 Constructor Method

The default constructor results in a enumeration literal list with no elements.

```
IIR_EnumerationLiteralList();
```

9.17.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The

remove_element() method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(             IIR_EnumerationLiteral*      element);

void
    append_element(              IIR_EnumerationLiteral*      element);
IR_Boolean
    insert_after_element(        IIR_EnumerationLiteral*      existing_element,
                                 IIR_EnumerationLiteral*      new_element);
IR_Boolean
    insert_before_element(       IIR_EnumerationLiteral*      existing_element,
                                 IIR_EnumerationLiteral*      new_element);
IR_Boolean
    remove_element(              IIR_EnumerationLiteral*      existing_element);
IIR_EnumerationLiteral*
    get_successor_element(       IIR_EnumerationLiteral*      existing_element);
IIR_EnumerationLiteral*
    get_predecessor_element(     IIR_EnumerationLiteral*      element);
IIR_EnumerationLiteral*
    get_first_element();
IIR_EnumerationLiteral*
    get_nth_element(             IR_Int32                     index);
IIR_EnumerationLiteral*
    get_last_element();
IR_Int32
    get_element_position(        IIR_EnumerationLiteral*      element);
```

9.17.3.3 List Destructor Method

The list destructor method deletes all enumeration literals within the enumeration literal list, then deletes the list object itself.

```
void
    ~IIR_EnumerationLiteralList();
```

9.18 IIR_GenericList

9.18.1 Derived Class Description

The predefined **IIR_GenericList** class represents ordered sets containing zero or more IIR_ConstantInterfaceDeclarations. Generic lists appear as predefined public data elements within IIR_EntityDeclarations, IIR_BlockStatements and IIR_ComponentDeclarations.

9.18.2 Properties

TABLE 77. IIR_GenericList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_GENERIC_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects appear as predefined public data elements within IIR_EntityDeclarations, IIR_BlockStatements4 and IIR_ComponentDeclarations
Application-specific data elements	Not allowed
Public data elements	None

9.18.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_GenericList object. All of the following methods are atomic.

9.18.3.1 Constructor Method

The default constructor results in a generic list with no generics.

```
IIR_GenericList();
```

9.18.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method

does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_ConstantInterfaceDeclaration*    element);

void
    append_element(             IIR_ConstantInterfaceDeclaration*    element);
IR_Boolean
    insert_after_element(       IIR_ConstantInterfaceDeclaration*    existing_element,
                                IIR_ConstantInterfaceDeclaration*    new_element);
IR_Boolean
    insert_before_element(      IIR_ConstantInterfaceDeclaration*    existing_element,
                                IIR_ConstantInterfaceDeclaration*    new_element);
IR_Boolean
    remove_element(             IIR_ConstantInterfaceDeclaration*    existing_element);
IIR_ConstantInterfaceDeclaration*
    get_successor_element(      IIR_ConstantInterfaceDeclaration*    existing_element);
IIR_ConstantInterfaceDeclaration*
    get_predecessor_element(    IIR_ConstantInterfaceDeclaration*    element);
IIR_ConstantInterfaceDeclaration*
    get_first_element();
IIR_ConstantInterfaceDeclaration*
    get_nth_element(            IR_Int32                             index);
IIR_ConstantInterfaceDeclaration*
    get_last_element();
IR_Int32
    get_element_position(       IIR_ConstantInterfaceDeclaration*    element);
```

9.18.3.3 List Destructor Method

The list destructor method deletes all generic declaration within the generic list, then deletes the list object itself.

```
void
    ~IIR_GenericList();
```

9.19 IIR_InterfaceList

9.19.1 Derived Class Description

The predefined **IIR_InterfaceList** class represents ordered sets containing zero or more IIR_InterfaceDeclarations. Interface class lists appear as predefined public data elements within IIR_SubprogramDeclarations.

9.19.2 PropertiesInterfaceList

TABLE 78. IIR_InterfaceList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_INTERFACE_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_InterfaceList class are predefined public data elements within IIR_SubprogramDeclarations.
Application-specific data elements	Not allowed
Public data elements	None

9.19.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_InterfaceList object. All of the following methods are atomic.

9.19.3.1 Constructor Method

The default constructor results in a interface list with no interface declarations.

```
IIR_InterfaceList();
```

9.19.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method

does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(              IIR_InterfaceDeclaration*     element);
void
    append_element(               IIR_InterfaceDeclaration*     element);
IR_Boolean
    insert_after_element(         IIR_InterfaceDeclaration*     existing_element,
                                  IIR_InterfaceDeclaration*     new_element);
IR_Boolean
    insert_before_element(        IIR_InterfaceDeclaration*     existing_element,
                                  IIR_InterfaceDeclaration*     new_element);
IR_Boolean
    remove_element(               IIR_InterfaceDeclaration*     existing_element);
IIR_InterfaceDeclaration*
    get_successor_element(        IIR_InterfaceDeclaration*     existing_element);
IIR_InterfaceDeclaration*
    get_predecessor_element(      IIR_InterfaceDeclaration*     element);
IIR_InterfaceDeclaration*
    get_first_element();
IIR_InterfaceDeclaration*
    get_nth_element(              IR_Int32                      index);
IIR_InterfaceDeclaration*
    get_last_element();
IR_Int32
    get_element_position(         IIR_InterfaceDeclaration*     element);
```

9.19.3.3 List Destructor Method

The list destructor method deletes all interface declarations within an interface list, then deletes the list object itself.

```
void
    ~IIR_InterfaceList();
```

9.20 IIR_LibraryUnitList

9.20.1 Derived Class Description

The predefined **IIR_LibraryUnitList** represents ordered sets containing zero or more IIR_LibraryUnits. These library unit lists appear as predefined public data elements within an IIR_DesignFile.

9.20.2 Properties

TABLE 79. IIR_LibraryUnitList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_LIBRARY_UNIT_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_LibraryUnitList class are predefined public data elements within an IIR_DesignFile
Application-specific data elements	Not allowed
Public data elements	None

9.20.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_LibraryUnitList object. All of the following methods are atomic.

9.20.3.1 Constructor Method

The default constructor results in a design unit list with no design units.

```
IIR_LibraryUnitList();
```

9.20.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method

returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(             IIR_LibraryUnit*      element);
void
    append_element(              IIR_LibraryUnit*      element);
IR_Boolean
    insert_after_element(        IIR_LibraryUnit*      existing_element,
                                 IIR_LibraryUnit*      new_element);
IR_Boolean
    insert_before_element(       IIR_LibraryUnit*      existing_element,
                                 IIR_LibraryUnit*      new_element);
IR_Boolean
    remove_element(              IIR_LibraryUnit*      existing_element);
IIR_LibraryUnit*
    get_successor_element(       IIR_LibraryUnit*      existing_element);
IIR_LibraryUnit*
    get_predecessor_element(     IIR_LibraryUnit*      element);
IIR_LibraryUnit*
    get_first_element();
IIR_LibraryUnit*
    get_nth_element(             IR_Int32              index);
IIR_LibraryUnit*
    get_last_element();
IR_Int32
    get_element_position(        IIR_LibraryUnit*      element);
```

9.20.3.3 List Destructor Method

The list destructor method deletes all design units within the library unit list, then deletes the library unit list object itself.

```
void
    ~IIR_LibraryUnitList();
```

9.21 IIR_PortList

9.21.1 Derived Class Description

The **IIR_PortList** class represents ordered sets containing zero or more IIR_SignalInterfaceDeclarations. Port list classes appear as predefined public data elements within IIR_EntityDeclarations, IIR_BlockStatements and IIR_ComponentDeclarations.

9.21.2 Properties

TABLE 80. IIR_PortList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_PORT_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, within the predefined IIR class hierarchy, objects of IIR_PortList class are predefined public data within IIR_EntityDeclarations, IIR_BlockStatements and IIR_ComponentDeclarations.
Application-specific data elements	Not allowed
Public data elements	None

9.21.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_PortList object. All of the following methods are atomic.

9.21.3.1 Constructor Method

The default constructor results in a port list with no ports.

```
IIR_PortList();
```

9.21.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The

remove_element() method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_SignalInterfaceDeclaration*      element);

void
    append_element(             IIR_SignalInterfaceDeclaration*      element);
IR_Boolean
    insert_after_element(       IIR_SignalInterfaceDeclaration*      existing_element,
                                IIR_SignalInterfaceDeclaration*      new_element);
IR_Boolean
    insert_before_element(      IIR_SignalInterfaceDeclaration*      existing_element,
                                IIR_SignalInterfaceDeclaration*      new_element);
IR_Boolean
    remove_element(             IIR_SignalInterfaceDeclaration*      existing_element);
IIR_SignalInterfaceDeclaration*
    get_successor_element(      IIR_SignalInterfaceDeclaration*      existing_element);
IIR_SignalInterfaceDeclaration*
    get_predecessor_element(    IIR_SignalInterfaceDeclaration*      element);
IIR_SignalInterfaceDeclaration*
    get_first_element();
IIR_SignalInterfaceDeclaration*
    get_nth_element(            IR_Int32                             index);
IIR_SignalInterfaceDeclaration*
    get_last_element();
IR_Int32
    get_element_position(       IIR_SignalInterfaceDeclaration*      element);
```

9.21.3.3 List Destructor Method

The list destructor method deletes all elements of the port list, then deletes the list object itself.

```
void
    ~IIR_PortList();
```

9.22 IIR_SelectedWaveformList

9.22.1 Derived Class Description

The predefined **IIR_SelectedWaveformList** class represents ordered sets containing zero or more IIR_SelectedWaveforms objects. Selected waveform lists appear as predefined public data elements within IIR_ConcurrentSelectedSignalAssignments.

9.22.2 Properties

TABLE 81. IIR_SelectedWaveformList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SELECTED_WAVEFORM_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_SelectedWaveformList class are predefined public data elements within IIR_ConcurrentSelectedSignalAssignments.
Application-specific data elements	Not allowed
Public data elements	None

9.22.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SelectedWaveformList object. All of the following methods are atomic.

9.22.3.1 Constructor Method

The default constructor results in a selected waveform list with no selected waveform elements.

```
IIR_SelectedWaveformList();
```

9.22.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method

does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_SelectedWaveform*        element);

void
    append_element(             IIR_SelectedWaveform*        element);
IR_Boolean
    insert_after_element(       IIR_SelectedWaveform*        existing_element,
                                IIR_SelectedWaveform*        new_element);
IR_Boolean
    insert_before_element(      IIR_SelectedWaveform*        existing_element,
                                IIR_SelectedWaveform*        new_element);
IR_Boolean
    remove_element(             IIR_SelectedWaveform*        existing_element);
IIR_SelectedWaveform*
    get_successor_element(      IIR_SelectedWaveform*        existing_element);
IIR_SelectedWaveform*
    get_predecessor_element(    IIR_SelectedWaveform*        element);
IIR_SelectedWaveform*
    get_first_element();
IIR_SelectedWaveform*
    get_nth_element(            IR_Int32                     index);
IIR_SelectedWaveform*
    get_last_element();
IR_Int32
    get_element_position(       IIR_SelectedWaveform*        element);
```

9.22.3.3 List Destructor Method

The list destructor method deletes all selected waveform elements in the selected waveform list, then deletes the list object itself.

```
void
    ~IIR_SelectedWaveformList();
```

9.23 IIR_SequentialStatementList

9.23.1 Derived Class Description

The predefined **IIR_SequentialStatementlLis**t class represents ordered sets containing zero or more IIR_SequentialStatements. Sequential statement lists appear as predefined public data directly or indirectly within IIR_ProcessStatements.

9.23.2 Properties

TABLE 82. IIR_SequentialStatementList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SEQUENTIAL_STATEMENT_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_SequentialStatementList class are predefined public data elements (directly or indirectly) within IIR_ProcessStatements.
Application-specific data elements	Not allowed
Public data elements	None

9.23.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SequentialStatementList object. All of the following methods are atomic.

9.23.3.1 Constructor Method

The default constructor results in a sequential statement list with no sequential statement elements.

```
IIR_SequentialStatementList();
```

9.23.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method

does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(              IIR_SequentialStatement*     element);
void
    append_element(               IIR_SequentialStatement*     element);
IR_Boolean
    insert_after_element(         IIR_SequentialStatement*     existing_element,
                                  IIR_SequentialStatement*     new_element);
IR_Boolean
    insert_before_element(        IIR_SequentialStatement*     existing_element,
                                  IIR_SequentialStatement*     new_element);
IR_Boolean
    remove_element(               IIR_SequentialStatement*     existing_element);
IIR_SequentialStatement*
    get_successor_element(        IIR_SequentialStatement*     existing_element);
IIR_SequentialStatement*
    get_predecessor_element(      IIR_SequentialStatement*     element);
IIR_SequentialStatement*
    get_first_element();
IIR_SequentialStatement*
    get_nth_element(              IR_Int32                     index);
IIR_SequentialStatement*
    get_last_element();
IR_Int32
    get_element_position(         IIR_SequentialStatement*     element);
```

9.23.3.3 List Destructor Method

The list destructor method deletes all sequential statements within a sequential statement list, then deletes the list object itself.

```
void
    ~IIR_SequentialStatementList();
```

9.24 IIR_SimultaneousAlternativeList

9.24.1 Derived Class Description

The **IIR_SimultaneousAlternativeList** class represents ordered sets containing zero or more **IIR_SimultaneousAlternatives**.

9.24.2 Properties

TABLE 83. **IIR_SimultaneousAlternativeList Properties**

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IIR_SIMULTANEOUS_ALTERNATIVE_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly as part of IIR_SimultaneousCaseStatement.
Application-specific data elements	Not allowed
Public data elements	None

9.24.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousAlternativeList object. All of the following methods are atomic.

9.24.3.1 Constructor Method

The default constructor results in a list with no elements.

```
IIR_SimultaneousAlternativeList();
```

9.24.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method

returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_SimultaneousAlternative*  element);
void
    append_element(             IIR_SimultaneousAlternative*  element);
IR_Boolean
    insert_after_element(       IIR_SimultaneousAlternative*  existing_element,
                                IIR_SimultaneousAlternative*  new_element);
IR_Boolean
    insert_before_element(      IIR_SimultaneousAlternative*  existing_element,
                                IIR_SimultaneousAlternative*  new_element);
IR_Boolean
    remove_element(             IIR_SimultaneousAlternative*  existing_element);
IIR_SimultaneousAlternative*
    get_successor_element(      IIR_SimultaneousAlternative*  existing_element);
IIR_SimultaneousAlternative*
    get_predecessor_element(    IIR_SimultaneousAlternative*  element);
IIR_SimultaneousAlternative*
    get_first_element();
IIR_SimultaneousAlternative*
    get_nth_element(            IR_Int32                      index);
IIR_SimultaneousAlternative*
    get_last_element();
IR_Int32
    get_element_position(       IIR_SimultaneousAlternative*  element);
```

9.24.3.3 List Destructor Method

The list destructor method deletes all elements of the simultaneous alternative list, then deletes the list object itself.

```
void
    ~IIR_SimultaneousAlternativeList();
```

9.25 IIR_SimultaneousStatementList

9.25.1 Derived Class Description

The **IIR_SimultaneousStatementList** class represents ordered sets containing zero or more **IIR_SimultaneousStatement**s.

9.25.2 Properties

TABLE 84. IIR_SimultaneousStatementList Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMULTANEOUS_STATEMENT_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly as part of simultaneous statements.
Application-specific data elements	Not allowed
Public data elements	None

9.25.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousStatementList object. All of the following methods are atomic.

9.25.3.1 Constructor Method

The default constructor results in a list with no elements.

```
IIR_SimultaneousStatementList();
```

9.25.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method

returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(             IIR_SimultaneousStatement*    element);
void
    append_element(              IIR_SimultaneousStatement*    element);
IR_Boolean
    insert_after_element(        IIR_SimultaneousStatement*    existing_element,
                                 IIR_SimultaneousStatement*    new_element);
IR_Boolean
    insert_before_element(       IIR_SimultaneousStatement*    existing_element,
                                 IIR_SimultaneousStatement*    new_element);
IR_Boolean
    remove_element(              IIR_SimultaneousStatement*    existing_element);
IIR_SimultaneousStatement*
    get_successor_element(       IIR_SimultaneousStatement*    existing_element);
IIR_SimultaneousStatement*
    get_predecessor_element(     IIR_SimultaneousStatement*    element);
IIR_SimultaneousStatement*
    get_first_element();
IIR_SimultaneousStatement*
    get_nth_element(             IR_Int32                      index);
IIR_SimultaneousStatement*
    get_last_element();
IR_Int32
    get_element_position(        IIR_SimultaneousStatement*    element);
```

9.25.3.3 List Destructor Method

The list destructor method deletes all elements of the break list, then deletes the list object itself.

```
void
    ~IIR_SimultaneousStatementList();
```

9.26 IIR_StatementList

9.26.1 Derived Class Description

The **IIR_StatementList** class represents ordered sets containing zero or more **IIR_SimultaneousStatements**.

9.26.2 Properties

TABLE 85. IIR_StatementList Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_STATEMENT_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	
Application-specific data elements	Not allowed
Public data elements	None

9.26.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_StatementList object. All of the following methods are atomic.

9.26.3.1 Constructor Method

The default constructor results in a list with no elements.

```
IIR_StatementList();
```

9.26.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***,

get_nth_element() and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(             IIR_Statement*          element);
void
    append_element(              IIR_Statement*          element);
IR_Boolean
    insert_after_element(         IIR_Statement*          existing_element,
                                  IIR_Statement*          new_element);
IR_Boolean
    insert_before_element(        IIR_Statement*          existing_element,
                                  IIR_Statement*          new_element);
IR_Boolean
    remove_element(               IIR_Statement*          existing_element);
IIR_Statement*
    get_successor_element(        IIR_Statement*          existing_element);
IIR_Statement*
    get_predecessor_element(      IIR_Statement*          element);
IIR_Statement*
    get_first_element();
IIR_Statement*
    get_nth_element(              IR_Int32                index);
IIR_Statement*
    get_last_element();
IR_Int32
    get_element_position(         IIR_Statement*          element);
```

9.26.3.3 List Destructor Method

The list destructor method deletes all elements of the break list, then deletes the list object itself.

```
void
    ~IIR_StatementList();
```

9.27 IIR_UnitList

9.27.1 Derived Class Description

The predefined **IIR_UnitList** class represents ordered sets containing zero or more IIR_PhysicalUnits. Unit lists appear as predefined public data within IIR_PhysicalTypeDefinition and IIR_PhysicalSubtypeDefinition classes.

9.27.2 Properties

TABLE 86. IIR_UnitList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_UNIT_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_UnitList class appear as predefined public data elements within IIR_PhysicalTypeDefinition and IIR_PhysicalSubtypeDefinition classes
Application-specific data elements	Not allowed
Public data elements	None

9.27.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_UnitList object. All of the following methods are atomic.

9.27.3.1 Constructor Method

The default constructor results in a unit list with no physical unit declarations.

```
IIR_UnitList();
```

9.27.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method

does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(             IIR_PhysicalUnit*     element);
void
    append_element(              IIR_PhysicalUnit*     element);
IR_Boolean
    insert_after_element(        IIR_PhysicalUnit*     existing_element,
                                 IIR_PhysicalUnit*     new_element);
IR_Boolean
    insert_before_element(       IIR_PhysicalUnit*     existing_element,
                                 IIR_PhysicalUnit*     new_element);
IR_Boolean
    remove_element(              IIR_PhysicalUnit*     existing_element);
IIR_PhysicalUnit*
    get_successor_element(       IIR_PhysicalUnit*     existing_element);
IIR_PhysicalUnit*
    get_predecessor_element(     IIR_PhysicalUnit*     element);
IIR_PhysicalUnit*
    get_first_element();
IIR_PhysicalUnit*
    get_nth_element(             IR_Int32              index);
IIR_PhysicalUnit*
    get_last_element();
IR_Int32
    get_element_position(        IIR_PhysicalUnit*     element);
```

9.27.3.3 List Destructor Method

The list destructor method deletes all physical units within the unit list, then deletes the list object itself.

```
void
    ~IIR_UnitList();
```

9.28 IIR_WaveformList

9.28.1 Derived Class Description

The predefined **IIR_WaveformList** represents ordered sets containing zero or more IIR_WaveformElements. Waveform lists appear as predefined public data within sequential and concurrent signal assignment statements.

9.28.2 Properties

TABLE 87. IIR_WaveformList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_WAVEFORM_LIST**
Parent class	**IIR_List**
Predefined child classes	None
Instantiation?	Indirectly, objects of IIR_WaveformList class are predefined public data elements (directly or indirectly) within sequential and concurrent signal assignment statements.
Application-specific data elements	Not allowed
Public data elements	None

9.28.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_WaveformList object. All of the following methods are atomic.

9.28.3.1 Constructor Method

The default constructor results in a waveform list with no waveform elements.

```
IIR_WaveformList();
```

9.28.3.2 Element Reference Methods

Element methods refer insert, access or remove an element of the list. The ***prepend_element()*** method inserts the specified element at the beginning of the list if and only if the element exists. The ***append_element()*** method appends the specified element as the last element of the list if and only if the element exists. The ***insert_after_element()*** method inserts the specified new_element after the existing_element. Conversely the ***insert_before_element()*** method inserts the specified new_element before the existing_element. The ***remove_element()*** method removes the specified element. If the existing_element of an insert or remove method

does not exist on the list, the method returns FALSE, otherwise it returns TRUE. If the new_element of an insert method does not exist, no insertion occurs (however the method returns TRUE). The ***get_successor()*** method returns the list element immediately after the existing_element (if one exists). The ***get_predecessor()*** method returns the list element immediately preceeding the existing element (if one exists). The ***get_first_element()***, ***get_nth_element()*** and ***get_last_element()*** return the specified element (if one exists). The ***get_element_position()*** returns an integer denoting the position of the specified element on the list. Position numbering begins with zero. If the element is not found, get_element_position returns -1.

```
void
    prepend_element(            IIR_WaveformElement*        element);
void
    append_element(             IIR_WaveformElement*        element);
IR_Boolean
    insert_after_element(       IIR_WaveformElement*        existing_element,
                                IIR_WaveformElement*        new_element);
IR_Boolean
    insert_before_element(      IIR_WaveformElement*        existing_element,
                                IIR_WaveformElement*        new_element);
IR_Boolean
    remove_element(             IIR_WaveformElement*        existing_element);
IIR_WaveformElement
    get_successor_element(      IIR_WaveformElement*        existing_element);
IIR_WaveformElement*
    get_predecessor_element(    IIR_WaveformElement*        element);
IIR_WaveformElement*
    get_first_element();
IIR_WaveformElement*
    get_nth_element(            IR_Int32                    index);
IIR_WaveformElement*
    get_last_element();
IR_Int32
    get_element_position(       IIR_WaveformElement*        element);
```

9.28.3.3 List Destructor Method

The list destructor method deletes all waveform elements of the waveform list, then deletes the list object itself.

```
void
    ~IIR_WaveformList();
```

CHAPTER 10

IIR_TypeDefinition and IIR_Nature Definition Derived Classes

Type definitions generally define a domain of allowable values and a set of operators on these values. Type declarations, subtype declarations, object instances and expressions refer to type definitions. This chapter specifies the properties, predefined public methods and predefined public data used to represent types via the predefined IIR_TypeDefinition class, predefined classes derived from IIR_TypeDefinition (as shown in Table 88 on page 193), the IIR_NatureDefinition class and predefined classes derived from IIR_NatureDefinition (as shown in Table 89 on page 194)

Value domains may be explicitly enumerated (as with enumerated types) and/or described as a range of allowable values (as with integer types). VHDL also provides physical types, whereby textually multipliers may simplify writing physical literals. One or more subtypes may be constructed from a base type through the application of type constraints.

Operator symbols are drawn from a predefined set. The behavior of an operator symbol in the context of parameters of a specific base type and specific return type is determined by the visible function declarations overloading the operator symbol (only VHDL allows operator overloading) and any predefined, implicit function declarations.

Type signatures extend the type concept to denote types associated with each parameter of a subprogram declaration and the subprogram's return type. Signatures are generally used to disambiguate among multiple subprogram declarations having the same declarator but different parameter and return type signatures.

All derivative classes of IIR_TypeDefinition and IIR_NatureDefinition are dynamically and individually allocated.

TABLE 88. **Classes derived from IIRBase_TypeDefinition**

	Level 3 Derived Classes (only classes derived from IIR_TypeDefinition)		**Level 4 Derived Classes**		**Level 5 Derived Classes**
E	IIR_ScalarTypeDefinition	E	*IIR_EnumerationTypeDefinition*	E	*IIR_EnumerationSubtypeDefinition*
		E	*IIR_IntegerTypeDefinition*	E	*IIR_IntegerSubtypeDefinition*

TABLE 88. Classes derived from IIRBase_TypeDefinition

		E	*IIR_FloatingTypeDefinition*	E	*IIR_FloatingSubtypeDefinition*
		E	*IIR_PhysicalTypeDefinition*	E	*IIR_PhysicalTypeDefinition*
		E	*IIR_RangeTypeDefinition*	E	
E	*IIR_ArrayTypeDefinition*	E	*IIR_ArraySubtypeDefinition*	E	
E	*IIR_RecordTypeDefinition*	E	*IIR_RecordSubtypeDefinition*	E	
E	*IIR_AccessTypeDefinition*	E	*IIR_AccessSubtypeDefinition*	E	
E	*IIR_FileTypeDefinition*	E		E	
E	*IIR_Signature*	E		E	

TABLE 89. Classes deriver from IIRBase_NatureDefinition

	Level 3 Derived Classes (only classes derived from IIR_NatureDefinition)		Level 4 Derived Classes		Level 5 Derived Classes
E	IIR_ScalarNatureDefinition	E			
E	IIR_CompositeNatureDefinition	E	IIR_ArrayNatureDefinition		IIR_ArraySubnatureDefinition
		E	IIR_RecordNatureDefinition		IIR_RecordSubnatureDefinition

10.1 IIR_TypeDefinition

10.1.1 Derived Class Description

IIR_TypeDefinitions represent predefined methods, subprograms and public data elements common to types, subtypes, subtype indications, natures, subnatures, and signatures.

10.1.2 Properties

TABLE 90. IIR_TypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR**
Predefined child classes	**IIR_ScalarTypeDefinition, IIR_ArrayTypeDefinition, IIR_RecordTypeDefinition, IIR_AccessTypeDefinition, IIR_FileTypeDefinition, IIR_Signature,**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

10.1.3 Predefined Public Methods

All IIR_TypeDefinition methods must be applied to a valid IIR_TypeDefinition and are atomic.

10.1.3.1 Base Type Methods

All type definitions have a base type, representing the broadest, unconstrained form of the type.

```
void
    set_base_type(IIR_TypeDefinition*    base_type);
IIR_TypeDefinition*
    get_base_type();
```

10.2 IIR_ScalarTypeDefinition

10.2.1 Derived Class Description

IIR_ScalarTypeDefinitions represent predefined methods, subprograms and public data elements common to enumerated type definitions, integer type definitions, floating point type definitions, physical type definitions and range type definitions.

10.2.2 Properties

TABLE 91. IIR_ScalarTypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_TypeDefinition**
Predefined child classes	**IIR_EnumerationTypeDefinition, IIR_IntegerTypeDefinition, IIR_FloatingTypeDefinition, IIR_PhysicalTypeDefinition, IIR_RangeTypeDefinition**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

10.2.3 Predefined Public Methods

All IIR_ScalarTypeDefinition methods must be applied to a valid IIR_ScalarTypeDefinition and are atomic.

10.2.3.1 Range Methods

All scalar types have limits. Base types have limits defining the maximum range of any allowable subtype of the base type. Subtypes denote the applicable constraints through range methods. An unconstrained scalar subtype has left and right limits which are NIL.

```
void
    set_left(             IIR*    left);
IIR*
    get_left();
void
    set_direction(        IIR*    direction);
IIR*
    get_direction();
void
    set_right(            IIR*    right);
```

```
IIR*
    get_right();
```

10.3 IIR_EnumerationTypeDefinition

10.3.1 Derived Class Description

The predefined **IIR_EnumerationTypeDefinition** represents its value domain by a set of enumeration literals.

10.3.2 Properties

TABLE 92. **IIR_EnumerationTypeDefinition Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_ENUMERATION_TYPE_DEFINITION**
Parent class	**IIR_ScalarTypeDefinition**
Predefined child classes	**IIR_EnumerationSubtypeDefinition**
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

10.3.3 Predefined Public Methods

IIR_EnumerationTypeDefinition methods other than the constructor must be applied to a valid IIR_EnumerationTypeDefinition. All of the following methods are atomic.

10.3.3.1 Constructor Method

The constructor method creates a valid enumerated type definition with no literals.

```
IIR_EnumerationTypeDefinition();
```

10.3.3.2 Destructor Method

The destructor method deletes each of the enumeration literals present in the enumerated type definition, then the enumerated type definition object itself.

```
~IIR_EnumerationTypeDefinition();
```

10.3.4 Predefined Public Data

The enumeration type definition has a single predefined, public data element representing the list of enumeration literals associated with the type definition.

```
IIR_EnumerationLiteralList        enumeration_literals;
```

10.4 IIR_EnumerationSubtypeDefinition

10.4.1 Derived Class Description

The predefined **IIR_EnumerationSubtypeDefinition** class represent a subset of the literals represented by an IIR_EnumerationTypeDefinition base type.

10.4.2 Properties

TABLE 93. IIR_EnumerationSubtypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_ENUMERATION_SUBTYPE_DEFINITION**
Parent class	**IIR_EnumerationTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes

10.4.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having IIR_EnumerationTypeDefinition class. All of the following methods are atomic.

10.4.3.1 Get Method

The get method returns the specified subtype of the designated base type. The get method may either call a constructor directly or use a canonical representation of the subtype.

```
static IIR_EnumerationSubtypeDefinition*
    get(    IIR_EnumerationTypeDefinition*        base_type,
            IIR_EnumerationLiteral*               left_limit,
            IIR_EnumerationLiteral*               right_limit,
            IIR_FunctionDeclaration*              resolution_function);
```

10.4.3.2 Base Type Methods

The derived type base method provides a means of acquiring the enumeration subtype's base type without casting.

```
void
    set_base_type(         IIR_EnumerationTypeDefinition*        base_type);
IIR_EnumerationTypeDefinition*
    get_base_type();
```

10.4.3.3 Resolution Function Methods

Resolution function methods optionally associate a function with the subtype which determines the value of a signal having the specified subtype and more than one driver.

```
void
   set_resolution_function(IIR_FunctionDeclaration*      resolution_function);
IIR_FunctionDeclaration*
   get_resolution_function();
```

10.4.3.4 Release Method

The release method releases the IIR_EnumerationSubtypeDefinition previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

10.4.4 Predefined Public Data

```
IIR_EnumerationLiteralList        enumeration_literals;
```

10.5 IIR_IntegerTypeDefinition

10.5.1 Derived Class Description

The predefined **IIR_IntegerTypeDefinition** class represents a range of integer point values.

10.5.2 Properties

TABLE 94. **IIR_IntegerTypeDefinition Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_INTEGER_TYPE_DEFINITION**
Parent class	**IIR_ScalarTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.5.3 Predefined Public Methods

IIR_IntegerTypeDefinition methods other than the constructor must be applied to a valid IIR_IntegerTypeDefinition. All of the following methods are atomic.

10.5.3.1 Constructor Method

The constructor method creates a valid integer type definition.

```
IIR_IntegerTypeDefinition();
```

10.5.3.2 Destructor Method

The destructor method deletes the left limit, direction and right limit, then the integer type definition object itself.

```
~IIR_IntegerTypeDefinition();
```

10.6 IIR_IntegerSubtypeDefinition

10.6.1 Derived Class Description

The predefined **IIR_IntegerSubtypeDefinition** class represents a subset of an existing integer base type definition.

10.6.2 Properties

TABLE 95. **IIR_IntegerSubtypeDefinition Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_INTEGER_SUBTYPE_DEFINITIO**N
Parent class	**IIR_IntegerTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.6.3 Predefined Public Methods

IIR_IntegerSubtypeDefinition methods other than the constructor must be applied to a valid IIR_IntegerSubtypeDefinition. All of the following methods are atomic.

10.6.3.1 Get Method

The get method returns the specified subtype of the designated base type. The get method may either call a constructor directly or use a canonical representation of the subtype.

```
static IIR_IntegerSubtypeDefinition*
    get(    IIR_IntegerTypeDefinition*    base_type,
            IIR*                          left_limit,
            IIR*                          direction,
            IIR*                          right_limit,
            IIR_FunctionDeclaration*      resolution_functio);
```

10.6.3.2 Base Type Methods

The derived type base method provides a means of acquiring the integer subtype's base type without casting.

10.6.3.3

```
void
set_base_type(IIR_IntegerTypeDefinition*base_type);
```

```
IIR_IntegerTypeDefinition*
                get_base_type();
```

10.6.3.4 Resolution Function Methods

Resolution function methods optionally associate a function with the subtype which determines the value of a signal having the specified subtype and more than one driver.

```
void
  set_resolution_function(IIR_FunctionDeclaration*     resolution_function);
IIR_FunctionDeclaration*
  get_resolution_function();
```

10.6.3.5 Release Method

The release method releases the IIR_IntegerSubtypeDefinition previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
   release();
```

10.7 IIR_FloatingTypeDefinition

10.7.1 Derived Class Description

The predefined **IIR_FloatingTypeDefinition** class represents a range of floating point values.

10.7.2 Properties

TABLE 96. IIR_FloatingTypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_FLOATING_TYPE_DEFINITION**
Parent class	**IIR_ScalarTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.7.3 Predefined Public Methods

IIR_FloatingTypeDefinition methods other than the constructor must be applied to a valid IIR_FloatingTypeDefinition. All of the following methods are atomic.

10.7.3.1 Constructor Method

The constructor method creates a valid floating type definition.

```
IIR_FloatingTypeDefinition();
```

10.7.3.2 Destructor Method

The destructor method deletes the left limit, direction and right limit, then the floating type definition object itself.

```
~IIR_FloatingTypeDefinition();
```

10.8 IIR_FloatingSubtypeDefinition

10.8.1 Derived Class Description

The predefined **IIR_FloatingSubtypeDefinitio**n class represents a subset of an existing floating base type definition.

10.8.2 Properties

TABLE 97. IIR_FloatingSubtypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IIR_FLOATING_SUBTYPE_DEFINITION**
Parent class	**IIR_FloatingTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.8.3 Predefined Public Methods

IIR_FloatingSubtypeDefinition methods other than the constructor must be applied to a valid IIR_FloatingSubtypeDefinition. All of the following methods are atomic.

10.8.3.1 Get Method

The get method returns the specified subtype of the designated base type. The get method may either call a constructor directly or use a canonical representation of the subtype.

```
static IIR_FloatingSubtypeDefinition*
    get(    IIR_FloatingTypeDefinition*   base_type,
            IIR*                          left_limit,
            IIR*                          direction,
            IIR*                          right_limit,
            IIR_FunctionDeclaration*      resolution_function);
```

10.8.3.2 Base Type Methods

The derived type base method provides a means of acquiring the floating subtype’s base type without casting.

```
void
    set_base_type( IIR_FloatingTypeDefinition*   base_type);
IIR_FloatingTypeDefinition*
    get_base_type();
```

10.8.3.3 Resolution Function Methods

Resolution function methods optionally associate a function with the subtype which determines the value of a signal having the specified subtype and more than one driver.

```
void
   set_resolution_function(IIR_FunctionDeclaration*     resolution_function);
IIR_FunctionDeclaration*
   get_resolution_function();
```

10.8.3.4 Release Method

The release method releases the IIR_FloatingSubtypeDefinition previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

10.9 IIR_PhysicalTypeDefinition

10.9.1 Derived Class Description

The predefined **IIR_PhysicalTypeDefinition** class represents a VHDL physical type, including limits, a primary unit and zero or more secondary units.

10.9.2 Properties

TABLE 98. **IIR_PhysicalTypeDefinition Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_PHYSICAL_TYPE_DEFINITION**
Parent class	**IIR_ScalarTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes

10.9.3 Predefined Public Methods

IIR_PhysicalTypeDefinition methods other than the constructor must be applied to a valid IIR_PhysicalTypeDefinition. All of the following methods are atomic.

10.9.3.1 Constructor Method

The constructor method creates a valid physical type definition.

```
IIR_PhysicalTypeDefinition();
```

10.9.3.2 Primary Unit Methods

Every VHDL physical type requires a primary unit. All other (secondary) units are multiples of the primary unit.

```
void
    set_primary_unit(      IIR_PhysicalUnit*      unit);
IIR_PhysicalUnit*
    get_primary_unit();
```

10.9.3.3 Destructor Method

The destructor method deletes the left limit, direction and right limit, then the physical type definition object itself

```
~IIR_PhysicalTypeDefinition();
```

10.9.4 Predefined Public Data

```
IIR_UnitList            units;
```

10.10 IIR_PhysicalSubtypeDefinition

10.10.1 Derived Class Description

The predefined **IIR_PhysicalSubtypeDefinition** class represents a subtype of an existing physical type. The subtype range must be a subset of the base type's range.

10.10.2 Properties

TABLE 99. IIR_PhysicalSubtypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_PHYSICAL_SUBTYPE_DEFINITION**
Parent class	**IIR_PhysicalTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.10.3 Predefined Public Methods

IIR_PhysicalSubtypeDefinition methods other than the constructor must be applied to a valid IIR_PhysicalSubtypeDefinition. All of the following methods are atomic.

10.10.3.1 Get Method

The get method returns the specified subtype of the designated base type. The get method may either call a constructor directly or use a canonical representation of the subtype.

```
static IIR_PhysicalSubtypeDefinition*
    get(    IIR_PhysicalTypeDefinition*   base_type,
            IIR*                          left_limit,
            IIR                           direction,
            IIR*                          right_limit,
            IIR_FunctionDeclaration*      resolution_function);
```

10.10.3.2 Base Type Methods

The derived type base method provides a means of acquiring the physical subtype's base type without casting.

```
void
    set_base_type(        IIR_PhysicalTypeDefinition*   base_type);
IIR_PhysicalTypeDefinition*
    get_base_type();
```

10.10.3.3 Resolution Function Methods

Resolution function methods optionally associate a function with the subtype which determines the value of a signal having the specified subtype and more than one driver.

```
void
   set_resolution_function(IIR_FunctionDeclaration*      resolution_function);
IIR_FunctionDeclaration*
   get_resolution_function();
```

10.10.3.4 Release Method

The release method releases the IIR_PhysicalSubtypeDefinition previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

10.11 IIR_RangeTypeDefinition

10.11.1 Derived Class Description

The predefined **IIR_RangeTypeDefinitio**n class represents discrete ranges which are not proper HDL types or subtypes.

10.11.2 Properties

TABLE 100. IIR_RangeTypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A, Verilog
IR_Kind enumeration value	**IR_RANGE_TYPE_DEFINITION**
Parent class	**IIR_ScalarTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.11.3 Predefined Public Methods

IIR_RangeTypeDefinition methods other than the constructor must be applied to a valid IIR_RangeTypeDefinition. All of the following methods are atomic.

10.11.3.1 Constructor Method

The constructor method creates a valid range type definition.

```
IIR_RangeTypeDefinition();
```

10.11.3.2 Destructor Method

The destructor method deletes the left limit, direction and right limit, then the range object itself.

```
~IIR_RangeTypeDefinition();
```

10.12 IIR_ArrayTypeDefinition

10.12.1 Derived Class Description

The predefined **IIR_ArrayTypeDefinition** class represents base types containing zero or more instances of the same elemental subtype. Multi-dimensional arrays and arrays include record (or other elements) may be represented using composite array elements. For example, the first dimension of a two-dimensional array would have an element which is itself an array. Array type definitions denote unconstrained steps in an array definition.

10.12.2 Properties

TABLE 101. IIR_ArrayTypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IIR_ARRAY_TYPE_DEFINITION**
Parent class	**IIR_TypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.12.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_ArrayTypeDefinition class. All of the following methods are atomic.

10.12.3.1 Constructor Method

The constructor method creates a valid array type definition.

```
IIR_ArrayTypeDefinition();
```

10.12.3.2 Index Subtype Methods

The index subtype methods refer to the array's index domain.

```
void
    set_index_subtype(    IIR_ScalarTypeDefinition*     index_subtype);
IIR_ScalarTypeDefinition*
    get_index_subtype();
```

10.12.3.3 Element Subtype Methods

The element subtype methods refer to the element subtype which is replicated to form the array.

```
void
    set_element_subtype(IIR_TypeDefinition*                    element_subtype);
IIR_TypeDefinition*
    get_element_subtype();
```

10.12.3.4 Destructor Method

The destructor method deletes the index and element subtypes, then the array object itself.

```
    ~IIR_ArrayTypeDefinition();
```

10.13 IIR_ArraySubtypeDefinition

10.13.1 Derived Class Description

The predefined **IIR_ArraySubtypeDefinition** class represents the subtype of a pre-existing array type definition; always a constrained step in an array type definition. This subtype has an array domain which is a subset of the base type's domain.

10.13.2 Properties

TABLE 102. IIR_ArraySubtypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_ARRAY_SUBTYPE_DEFINITION**
Parent class	**IIR_ArrayTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.13.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_ArraySubtypeDefinition class. All of the following methods are atomic.

10.13.3.1 Get Method

The get method returns the specified subtype of the designated base type. The get method may either call a constructor directly or use a canonical representation of the subtype.

```
static IIR_ArraySubtypeDefinition*
    get(    IIR_ArrayTypeDefinition*      base_type,
            IIR_ScalarTypeDefinition*     index_subtype,
            IIR_FunctionDeclaration*      resolution_function);
```

10.13.3.2 Base Type Methods

The derived type base method provides a means of acquiring the array subtype's base type without casting.

```
void
    set_base_type(IIR_ArrayTypeDefinition*      base_type);
IIR_ArrayTypeDefinition*
    get_base_type();
```

10.13.3.3 Resolution Function Methods

Resolution function methods optionally associate a function with the subtype which determines the value of a signal having the specified subtype and more than one driver.

```
void
   set_resolution_function(IIR_FunctionDeclaration*     resolution_function);
IIR_FunctionDeclaration*
    get_resolution_function();
```

10.13.3.4 Release Method

The release method releases the IIR_ArraySubtypeDefinition previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

10.14 IIR_RecordTypeDefinition

10.14.1 Derived Class Description

The predefined **IIR_RecordTypeDefinition** class represents a record type having zero or more element declarations.

10.14.2 Properties

TABLE 103. IIR_RecordTypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_RECORD_TYPE_DEFINITION**
Parent class	**IIR_TypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes

10.14.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_RecordTypeDefinition class. All of the following methods are atomic.

10.14.3.1 Constructor Method

The constructor method creates a valid record type definition without element declarations.

```
IIR_RecordTypeDefinition();
```

10.14.3.2 Destructor Method

The destructor method deletes the record element declarations, then the record object itself.

```
~IIR_RecordTypeDefinition();
```

10.14.4 Predefined Public Data

```
IIR_ElementDeclarationList        element_declarations;
```

10.15 IIR_RecordSubtypeDefinition

10.15.1 Derived Class Description

The predefined **IIR_RecordSubtypeDefinition** class represents the subtype of a pre-existing record type definition; always a constrained step in an record type definition. This subtype has an record domain which is a subset of the base type's domain.

10.15.2 Properties

TABLE 104. **IIR_RecordSubtypeDefinition Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_RECORD_SUBTYPE_DEFINITION**
Parent class	**IIR_RecordTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.15.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_RecordSubtypeDefinition class. All of the following methods are atomic.

10.15.3.1 Get Method

The get method returns the specified subtype of the designated base type. The get method may either call a constructor directly or use a canonical representation of the subtype.

```
static IIR_RecordSubtypeDefinition*
    get(    IIR_RecordTypeDefinition*     base_type,
            IIR_FunctionDeclaration*      resolution_function);
```

10.15.3.2 Base Type Methods

The derived type base method provides a means of acquiring the Record subtype's base type without casting.

```
void
    set_base_type(IIR_RecordTypeDefinition*      base_type);
IIR_RecordTypeDefinition*
    get_base_type();
```

10.15.3.3 Resolution Function Methods

Resolution function methods optionally associate a function with the subtype which determines the value of a signal having the specified subtype and more than one driver.

```
void
    set_resolution_function(IIR_FunctionDeclaration*     resolution_function);
IIR_FunctionDeclaration*
    get_resolution_function();
```

10.15.3.4 Release Method

The release method releases the IIR_RecordSubtypeDefinition previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

10.16 IIR_AccessTypeDefinition

10.16.1 Derived Class Description

The predefined **IIR_AccessTypeDefinition** class represents a type definition denoting a dynamically allocated object of designated type.

10.16.2 Properties

TABLE 105. IIR_AccessTypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_ACCESS_TYPE_DEFINITION**
Parent class	**IIR_TypeDefinition**
Predefined child classes	**IIR_AccessSubtypeDefinition**
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.16.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_AccessTypeDefinition class. All of the following methods are atomic.

10.16.3.1 Get Method

The get method creates a valid access type definition for the designated type.

```
IIR_AccessTypeDefinition*
    get(    IIR_TypeDefinition*                     designated_type);
```

10.16.3.2 Designated Subtype Methods

The designated type methods denote the type to which this access type refers.

```
void
    set_designated_type(   IIR_TypeDefinition*    designated_type);
IIR_TypeDefinition*
    get_designated_type();
```

10.16.3.3 Destructor Method

The destructor method deletes the access type definition object itself.

```
~IIR_AccessTypeDefinition();
```

10.17 IIR_AccessSubtypeDefinition

10.17.1 Derived Class Description

The predefined IIR_AccessSubtypeDefinition class represents a subtype of an access type definition.

10.17.2 Properties

TABLE 106. IIR_AccessSubtypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IIR_ACCESS_SUBTYPE_DEFINITION**
Parent class	**IIR_AccessTypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.17.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_AccessSubtypeDefinition class. All of the following methods are atomic.

10.17.3.1 Get Method

The get method returns the specified subtype of the designated base type. The get method may either call a constructor directly or use a canonical representation of the subtype.

```
static IIR_AccessSubtypeDefinition*
    get(          IIR_TypeDefinition*          designated_type,
                  IIR_FunctionDeclaration*     resolution_function);
```

10.17.3.2 Designated Subtype Methods

The designated subtype methods denote the subtype to which this access subtype refers.

```
void
    set_designated_subtype(       IIR_TypeDefinition*    designated_type);
IIR_TypeDefinition*
    get_designated_subtype();
```

10.17.3.3 Resolution Function Methods

Resolution function methods optionally associate a function with the subtype which determines the value of a signal having the specified subtype and more than one driver.

```
void
   set_resolution_function(IIR_FunctionDeclaration*     resolution_function);
IIR_FunctionDeclaration*
   get_resolution_function();
```

10.17.3.4 Release Method

The release method releases the IIR_AccessSubtypeDefinition previously acquired through a get. If the get is implemented via a constructor, the release method should generally be implemented via a destructor; generally each get and release must be matched.

```
void
    release();
```

10.18 IIR_FileTypeDefinition

10.18.1 Derived Class Description

The predefined **IIR_FileTypeDefinitio**n represents the type associated with zero or more file declarations.

10.18.2 Properties

TABLE 107. IIR_FileTypeDefinition Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_FILE_TYPE_DEFINITION**
Parent class	**IIR_TypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.18.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_FileTypeDefinition.   All of the following methods are atomic.

10.18.3.1 Constructor Method

The constructor method creates a valid file type definition with an undefined type mark.

```
IIR_FileTypeDefinition();
```

10.18.3.2 Type Mark Methods

The type mark methods refer to the type of data which is stored in files having this file type definition.

```
void
    set_type_mark(        IIR_TypeDefinition*    type_mark);
IIR_TypeDefinition*
    get_type_mark();
```

10.18.3.3 Destructor Method

The destructor deletes the file type object.

```
~IIR_FileTypeDefinition();
```

10.19 IIR_Signature

10.19.1 Derived Class Description

The predefined **IIR_Signature** class represents the parameter type signature and optional return type which may be associated with one or more subprograms. Note that the signature does not directly imply interface declarators or a subprogram declarator.

10.19.2 Properties

TABLE 108. IIR_Signature Properties

Applicable language(s)	VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_SIGNATURE**
Parent class	**IIR_TypeDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes

10.19.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_Signature class. All of the following methods are atomic.

10.19.3.1 Constructor Method

The constructor creates a valid signature object with no argument types and no return type.

```
IIR_Signature();
```

10.19.3.2 Return Type Methods

The return type methods refers to the return type of the associated signature.

```
void
    set_return_type(       IIR_TypeDefinition*    return_type);
IIR_TypeDefinition*
    get_return_type();
```

10.19.3.3 Destructor Method

The destructor deletes an object of IIR_Signature class.

```
~IIR_Signature();
```

10.19.4 Predefined Public Data

The IIR_Signature class has a single public data element:

```
IIR_DesignatorList        argument_type_list;
```

10.20 IIR_NatureDefinition

10.20.1 Derived Class Description

The **IIR_NatureDefinition** class represent predefined methods, subprograms and public data elements describing scalar natures.

10.20.2 Properties

TABLE 109. IIR_ScalarNatureDefinition Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_NATURE_DEFINITION**
Parent class	**IIR_TypeDefinition**
Predefined child classes	**IIR_ScalarNatureDefinition** **IIR_CompositeNatureDefinition**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

10.20.3 Predefined Public Methods

All IIR_NatureDefinition methods must be applied to a valid IIR_NatureDefinition and are atomic.

10.21 IIR_ScalarNatureDefinition

10.21.1 Derived Class Description

The **IIR_ScalarNatureDefinition** class represent predefined methods, subprograms and public data elements describing scalar natures.

10.21.2 Properties

TABLE 110. IIR_ScalarNatureDefinition Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SCALAR_NATURE_DEFINITION**
Parent class	**IIR_NatureDefinition**
Predefined child classes	**IIR_ScalarSubnatureDefinition**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

10.21.3 Predefined Public Methods

All IIR_ScalarNatureDefinition methods must be applied to a valid IIR_ScalarNatureDefinition and are atomic.

10.21.3.1 Constructor Method

```
IIR_ScalarNatureDefinition();
```

10.21.3.2 Across Type Methods

```
void
    set_across(IIR_NatureDefinition*            across);
IIR_NatureDefinition*
    get_across();
```

10.21.3.3 Through Type Methods

```
void
    set_through(IIR_NatureDefinition*            through);
IIR_NatureDefinition*
    get_through();
```

10.21.3.4 Destructor Method

The destructor deletes an object of IIR_ScalarNatureDefinition class.

```
    ~IIR_ScalarNatureDefinition();
```

10.22 IIR_ScalarSubatureDefinition

10.22.1 Derived Class Description

The **IIR_ScalarSubnatureDefinition** class represent predefined methods, subprograms and public data elements describing scalar subnatures.

10.22.2 Properties

TABLE 111. IIR_ScalarNatureDefinition Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SCALAR_SUBNATURE_DEFINITION**
Parent class	**IIR_ScalarNatureDefinition**
Predefined child classes	None
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

10.22.3 Predefined Public Methods

All IIR_ScalarSubnatureDefinition methods must be applied to a valid IIR_ScalarSubnatureDefinition and are atomic.

10.22.3.1 Constructor Method

```
IIR_ScalarSubnatureDefinition();
```

10.22.3.2 Tolerance Methods

```
void
    set_across_tolerance(IIR*                    across_tolerance);
IIR*
    get_across_tolerance();
void
    set_through_tolerance(IIR*                   through_tolerance);
IIR*
    get_through_tolerance();
```

10.22.3.3 Base Nature Methods

```
IIR_ArrayNatureDefinition*
    get_base_nature();
```

10.22.3.4 Destructor Method

The destructor deletes an object of IIR_ScalarNatureDefinition class.

```
    ~IIR_ScalarSubnatureDefinition();
```

10.23 IIR_CompositeNatureDefinition

10.23.1 Derived Class Description

The **IIR_CompositeNatureDefinition** class represent predefined methods, subprograms and public data elements describing composite natures.

10.23.2 Properties

TABLE 112. IIR_ScalarNatureDefinition Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**None, not directly instantiated**
Parent class	**IIR_NatureDefinition**
Predefined child classes	**IIR_ArrayNatureDefinition** **IIR_RecordNatureDefinition**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

10.23.3 Predefined Public Methods

All IIR_CompositeNatureDefinition methods must be applied to a valid IIR_ScalarNatureDefinition and are atomic.

10.24 IIR_ArrayNatureDefinition

10.24.1 Derived Class Description

The predefined **IIR_ArrayNatureDefinition** class represents natures containing zero or more instances of the same elemental subtype. Multi-dimensional arrays and arrays include record (or other elements) may be represented using composite array elements. For example, the first dimension of a two-dimensional array would have an element which is itself an array.

10.24.2 Properties

TABLE 113. **IIR_ArrayNatureDefinition Properties**

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IIR_ARRAY_NATURE_DEFINITION**
Parent class	**IIR_CompositeNatureDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Public data elements	None

10.24.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_ArrayNatureDefinition class. All of the following methods are atomic.

10.24.3.1 Constructor Method

The constructor method creates a valid array type definition.

```
    IIR_ArrayNatureDefinition();
```

10.24.3.2 Index Subtype Methods

The index subtype methods refer to the array's index domain.

```
void
    set_index_subtype(    IIR_ScalarTypeDefinition*     index_subtype);
IIR_ScalarTypeDefinition*
    get_index_subtype();
```

10.24.3.3 Element Subtype Methods

The element subtype methods refer to the element subtype which is replicated to form the array. The element subtype must be a scalar or composite nature.

```
void
    set_element_subtype(IIR_NatureDefinition*                  element_subtype);
IIR_NatureDefinition*
    get_element_subtype();
```

10.24.3.4 Destructor Method

The destructor method deletes the index and element subtypes, then the array object itself.

```
    ~IIR_ArrayNatureDefinition();
```

10.25 IIR_ArraySubnatureDefinition

10.25.1 Derived Class Description

The **IIR_ArraySubnatureDefinition** class represent predefined methods, subprograms and public data elements describing array subnatures.

10.25.2 Properties

TABLE 114. IIR_ArraySubnatureDefinition Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_ARRAY_SUBNATURE_DEFINITION**
Parent class	**IIR_ArrayNatureDefinition**
Predefined child classes	None
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

10.25.3 Predefined Public Methods

All IIR_ArraySubnatureDefinition methods must be applied to a valid IIR_ArraySubnatureDefinition and are atomic.

10.25.3.1 Constructor Method

```
IIR_ArraySubnatureDefinition();
```

10.25.3.2 Tolerance Methods

```
void
    set_across_tolerance(IIR*                    across_tolerance);
IIR*
    get_across_tolerance();
void
    set_through_tolerance(IIR*                   through_tolerance);
IIR*
    get_through_tolerance();
```

10.25.3.3 Base Nature Methods

```
IIR_ArrayNatureDefinition*
    get_base_nature();
```

10.25.3.4 Destructor Method

The destructor deletes an object of IIR_ArraySubnatureDefinition class.

```
    ~IIR_ArraySubnatureDefinition();
```

10.26 IIR_RecordNatureDefinition

10.26.1 Derived Class Description

The predefined **IIR_RecordTypeDefinition** class represents a record nature having zero or more element declarations.

10.26.2 Properties

TABLE 115. IIR_RecordNatureDefinition Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_RECORD_NATURE_DEFINITION**
Parent class	**IIR_CompositeNatureDefinition**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes

10.26.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object of IIR_RecordNatureDefinition class. All of the following methods are atomic.

10.26.3.1 Constructor Method

The constructor method creates a valid record nature definition without element declarations.

```
IIR_RecordTNatureDefinition();
```

10.26.3.2 Destructor Method

The destructor method deletes the record element declarations, then the record object itself.

```
~IIR_RecordTNatureDefinition();
```

10.26.4 Predefined Public Data

```
IIR_ElementDeclarationList        element_declarations;
```

10.27 IIR_RecordSubnatureDefinition

10.27.1 Derived Class Description

The **IIR_RecordSubnatureDefinition** class represent predefined methods, subprograms and public data elements describing record subnatures.

10.27.2 Properties

TABLE 116. IIR_RecordSubnatureDefinition Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_RECORD_SUBNATURE_DEFINITION**
Parent class	**IIR_RecordNatureDefinition**
Predefined child classes	None
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

10.27.3 Predefined Public Methods

All IIR_RecordSubnatureDefinition methods must be applied to a valid IIR_RecordSubnatureDefinition and are atomic.

10.27.3.1 Constructor Method

```
IIR_RecordSubnatureDefinition();
```

10.27.3.2 Tolerance Methods

```
void
    set_across_tolerance(IIR*                    across_tolerance);
IIR*
    get_across_tolerance();
void
    set_through_tolerance(IIR*                   through_tolerance);
IIR*
    get_through_tolerance();
```

10.27.3.3 Base Nature Methods

```
IIR_RecordNatureDefinition*
    get_base_nature();
```

10.27.3.4 Destructor Method

The destructor deletes an object of IIR_RecordSubnatureDefinition class.

```
    ~IIR_RecordSubnatureDefinition();
```

CHAPTER 11

IIR_Declaration Derived Classes

Declarations introduced named entities or specifications into a declarative region. All such declarations and specifications are descended from IIR_Declaration, as shown in Table 117 on page 241. All classes derived from IIR_Declaration are directly and dynamically instantiated (rather than being incorporated in other classes).

TABLE 117. Class hierarchy derived from IIR_Declaration

	Level 3 Derived Classes (only classes derived from IIR_Declaration)		Level 4 Derived Classes		Level 5 Derived Classes
E	IIR_SubprogramDeclaration	E	*IIR_FunctionDeclaration*		
		E	*IIR_ProcedureDeclaration*		
E	*IIR_EnumerationLiteral*				
E	*IIR_ElementDeclaration*				
E	*IIR_NatureElementDeclaration*				
E	*IIR_TypeDeclaration*				
E	*IIR_SubtypeDeclaration*				
E	*IIR_NatureDeclaration*				
E	*IIR_SubnatureDeclaration*				
E	IIR_ObjectDeclaration	E	*IIR_ConstantDeclaration*		
		E	*IIR_VariableDeclaration*		
			IIR_SharedVariableDeclaration		
		E	*IIR_SignalDeclaration*		
		E	*IIR_FileDeclaration*		
		E	*IIR_TerminalDeclaration*		
		E	IIR_QuantityDeclaration	E	IIR_FreeQuantityDeclaration
				E	IIR_AcrossQuantityDeclaration

TABLE 117. Class hierarchy derived from IIR_Declaration

				E	IIR_NoiseSourceQuantityDeclaration
				E	IIR_SpectrumSourceQuantityDeclaration
				E	IIR_ThroughQuantityDeclaration
E	IIR_InterfaceDeclaration	E	*IIR_ConstantInterfaceDeclaration*		
		E	*IIR_VariableInterfaceDeclaration*		
		E	*IIR_SignalInterfaceDeclaration*		
		E	*IIR_FileInterfaceDeclaration*		
		E	*IIR_TerminalInterfaceDeclaration*		
		E	*IIR_QuantityInterfaceDeclaration*		
E	*IIR_AliasDeclaration*				
E	*IIR_AttributeDeclaration*				
E	*IIR_ComponentDeclaration*				
E	*IIR_GroupDeclaration*				
E	*IIR_GroupTemplateDeclaration*				
E	*IIR_LibraryDeclaration*				
E	IIR_LibraryUnit	E	*IIR_EntityDeclaration*		
		E	*IIR_ArchitectureDeclaration*		
		E	*IIR_PackageDeclaration*		
		E	*IIR_PackageBodyDeclaration*		
		E	*IIR_ConfigurationDeclaration*		
E	*IIR_PhysicalUnit*				
E	*IIR_AttributeSpecification*				
E	*IIR_ConfigurationSpecification*				
E	*IIR_DisconnectionSpecification*				
E	*IIR_Label*				
E	*IIR_LibraryClause*				
E	*IIR_UseClause*				

11.1 IIR_Declaration

11.1.1 Derived Class Description

The predefined IIR_Declaration class is the parent for all predefined declarations, specifications, library clauses or use clauses.

11.1.2 Properties

TABLE 118. IIR_Declaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None, not directly instantiated
Parent class	IIR
Predefined child classes	IIR_SubprogramDeclaration IIR_EnumerationLiteral IIR_ElementDeclaration IIR_NatureElementDeclaration IIR_TypeDeclaration IIR_SubtypeDeclaration IIR_NatureDeclaration IIR_SubnatureDeclaration IIR_ObjectDeclaration IIR_InterfaceDeclaration IIR_AliasDeclaration IIR_AttributeDeclaration IIR_ComponentDeclaration IIR_GroupDeclaration IIR_GroupTemplateDeclaration IIR_LibraryDeclaration IIR_LibraryUnit IIR_PhysicalUnit IIR_AttributeSpecification IIR_ConfigurationSpecification IIR_DisconnectionSpecification IIR_Label IIR_LibraryClause IIR_UseClause
Instantiation?	No, not directly from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

11.1.3 Predefined Public Methods

All IIR_Declaration methods must be applied to a valid IIR_Declaration class and are atomic.

11.1.3.1 Declarator Methods

Declarator methods refer to an IIR_TextLiteral (generally an IIR_Identifier) which is the declarator (declaration), an attribute designator (attribute specification), a component identifier (configuration specification), guarded signal name (disconnect specification), logical name (library clause) or selected name (use clause). Some IIR_Declaration classes do not have declarators, for example an IIR_DisconnectionSpecification has no declarator.

```
void
    set_declarator(IIR_TextLiteral*       identifier);
IIR_TextLiteral*
    get_declarator();
```

11.2 IIR_SubprogramDeclaration

11.2.1 Derived Class Description

The predefined **IIR_SubprogramDeclaration** class represents declarations and sequential code fragments which are dynamically elaborated when encountered at a procedure or function call site.

11.2.2 Properties

TABLE 119. IIR_SubprogramDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Declaration**
Predefined child classes	**IIR_ProcedureDeclaration** **IIR_FunctionDeclaration**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class

11.2.3 Predefined Public Methods

All IIR_SubprogramDeclarations methods must be applied to a valid IIR_SubprogramDeclaration class and are atomic.

11.2.4 Predefined Public Data Elements

The predefined public data elements consist of four lists: interface declarations, subprogram declarations, sequential statements within the subprogram body and attributes which may be associated with the subprogram declaration.

```
IIR_InterfaceList                  interface_declarations;
IIR_DeclarationList                subprogram_declarations;
IIR_SequentialStatementList        subprogram_body;
IIR_AttributeSpecificationList     attributes;
```

11.3 IIR_ProcedureDeclaration

11.3.1 Derived Class Description

The **IIR_ProcedureDeclaration** class represents subprograms callable with in, out, or inout parameters but without a return value.

11.3.2 Properties

TABLE 120. IIR_ProcedureDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98, VHDL-AMS
IR_Kind enumeration value	**IIR_PROCEDURE_DECLARATION**
Parent class	**IIR_SubprogramDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Predefined public data elements	None

11.3.3 Predefined Public Methods

IIR_ProcedureDeclaration methods other than the constructor must be applied to a valid IIR_ProcedureDeclaration. All of the following methods are atomic.

11.3.3.1 Constructor Method

The constructor method initializes a subprogram declaration with an unspecified location, an unspecified declarator, no interface declarations, no subprogram declarations and no subprogram statements.

```
IIR_ProcedureDeclaration();
```

11.3.3.2 Destructor Method

The destructor method each of the interface declarations, subprogram declarations and subprogram statements, then the deletes the procedure declaration object itself.

```
~IIR_ProcedureDeclaration();
```

11.4 IIR_FunctionDeclaration

11.4.1 Derived Class Description

The **IIR_FunctionDeclaration** class represents subprograms callable with in, out, or inout parameters with a return value.

11.4.2 Properties

TABLE 121. IIR_FunctionDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_FUNCTION_DECLARATION**
Parent class	**IIR_SubprogramDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Predefined public data elements	None

11.4.3 Predefined Public Methods

IIR_FunctionDeclaration methods other than the constructor must be applied to a valid IIR_FunctionDeclaration. All of the following methods are atomic.

11.4.3.1 Constructor Method

The constructor method initializes a subprogram declaration with an unspecified source location, an unspecified declarator, an unspecified return type, no interface declarations, no subprogram declarations and no subprogram statements.

```
    IIR_FunctionDeclaration();
```

11.4.3.2 Pure Method

```
void
    set_pure(IIR_Pure      purity);
IIR_Pure
    get_pure();
```

11.4.3.3 Return Type Methods

The return type methods reference the function declaration's return type.

```
void
    set_return_type(IIR_TypeDefinition*          return_type);
IIR_TypeDefinition*
    get_return_type();
```

11.4.3.4 Destructor Method

The destructor method each of the interface declarations, subprogram declarations and subprogram statements, then the deletes the function declaration object itself.

```
~IIR_FunctionDeclaration();
```

11.5 IIR_EnumerationLiteral

11.5.1 Derived Class Description

The predefined **IIR_EnumerationLiteral** represents a literal within an enumerated type or subtype definition.

11.5.2 Properties

TABLE 122. IIR_EnumerationLiteral Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ENUMERATION_LITERAL**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.5.3 Predefined Public Methods

IIR_EnumerationLiteral methods other than the constructor must be applied to a valid IIR_EnumerationLiteral. All of the following methods are atomic.

11.5.3.1 Constructor Method

The constructor method initializes an enumeration literal with an unspecified source location, an unspecified declarator, an unspecified position, and an unspecified enumeration subtype.

```
        IIR_EnumerationLiteral();
```

11.5.3.2 Position Methods

Position methods refer to the position of an enumeration literal within an enumeration type definition.

```
    void
        set_position(  IIR*            position);
    IIR*
        get_position();
```

11.5.3.3 Subtype Methods

Subtype Methods refer to the subtype which the enumeration literal is a part of.

```
void
    set_subtype(   IIR_EnumerationTypeDefinition*       subtype);
IIR_EnuemrationTypeDefinition*
    get_subtype();
```

11.5.3.4 Destructor Method

The destructor method deletes the enumerated literal object itself.

```
    ~IIR_EnumerationLiteral();
```

11.5.4 Predefined Public Data Elements

The predefined public data elements consist of a list of attributes which may be associated with the literal declaration.

```
IIR_AttributeSpecificationList    attributes;
```

11.6 IIR_ElementDeclaration

11.6.1 Derived Class Description

The predefined **IIR_ElementDeclaration** represents a element field within a record type definition.

11.6.2 Properties

TABLE 123. **IIR_ElementDeclaration Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ELEMENT_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.6.3 Predefined Public Methods

IIR_ElementDeclaration methods other than the constructor must be applied to a valid IIR_ElementDeclaration. All of the following methods are atomic.

11.6.3.1 Constructor Method

The constructor method initializes an element declaration with an unspecified source location, an unspecified declarator, and an unspecified element (sub)type.

```
IIR_ElementDeclaration();
```

11.6.3.2 Subtype Methods

Subtype Methods refer to the subtype of the record element itself.

```
void
    set_subtype(IIR_TypeDefinition*       subtype);
IIR_TypeDefinition*
    get_subtype();
```

11.6.3.3 Destructor Method

The destructor method deletes the element declaration object itself.

```
~IIR_ElementDeclaration();
```

11.7 IIR_NatureElementDeclaration

11.7.1 Derived Class Description

The predefined **IIR_NatureElementDeclaration** represents a element field within a record type definition.

11.7.2 Properties

TABLE 124. IIR_NatureElementDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_NATURE_ELEMENT_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.7.3 Predefined Public Methods

IIR_NatureElementDeclaration methods other than the constructor must be applied to a valid IIR_NatureElementDeclaration. All of the following methods are atomic.

11.7.3.1 Constructor Method

The constructor method initializes an element declaration with an unspecified source location, an unspecified declarator, and an unspecified element (sub)type.

```
IIR_NatureElementDeclaration();
```

11.7.3.2 Subnature Methods

Subnature Methods refer to the subnature of the nature record element itself.

```
void
    set_subnature(IIR_NatureDefinition*  subtype);
IIR_NatureDefinition*
    get_subnature();
```

11.7.3.3 Destructor Method

The destructor method deletes the nature element declaration object itself.

```
~IIR_NatureElementDeclaration();
```

11.8 IIR_TypeDeclaration

11.8.1 Derived Class Description

The predefined **IIR_TypeDeclaration** class represents the explicit declaration of a new type definition.

11.8.2 Properties

TABLE 125. **IIR_TypeDeclaration Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_TYPE_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.8.3 Predefined Public Methods

IIR_TypeDeclaration methods other than the constructor must be applied to a valid IIR_TypeDeclaration. All of the following methods are atomic.

11.8.3.1 Constructor Method

The constructor method initializes a type declaration with an unspecified source location, an unspecified declarator and unspecified type definition (forward declaration of type).

```
        IIR_TypeDeclaration();
```

11.8.3.2 Type Definition Methods

Type definition methods associate a separately constructed type definition with the type declaration

```
    void
        set_type(IIR_TypeDefinition*                  type);
    IIR_TypeDefinition*
        get_type();
```

11.8.3.3 Destructor Method

The destructor method deletes the type definition before deleting the type declaration itself.

```
~IIR_TypeDeclaration();
```

11.8.4 Predefined Public Data Elements

The predefined public data elements consist of a list of attributes which may be associated with the type declaration.

```
IIR_AttributeSpecificationList          attributes;
```

11.9 IIR_SubtypeDeclaration

11.9.1 Derived Class Description

The predefined **IIR_SubtypeDeclaration** class represents the explicit declaration of a new subtype definition.

11.9.2 Properties

TABLE 126. IIR_SubtypeDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_SUBTYPE_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.9.3 Predefined Public Methods

IIR_SubtypeDeclaration methods other than the constructor must be applied to a valid IIR_SubtypeDeclaration. All of the following methods are atomic.

11.9.3.1 Constructor Method

The constructor method initializes a type declaration with an unspecified source location, an unspecified declarator and unspecified subtype definition (forward declaration of a subtype).

```
        IIR_SubtypeDeclaration();
```

11.9.3.2 Subtype Definition Methods

Subtype definition methods associate a separately constructed subtype definition with the subtype declaration

```
    void
        set_subtype (IIR_TypeDefinition*      subtype);
    IIR_SubtypeDefinition*
        get_subtype();
```

11.9.3.3 Destructor Method

The destructor method deletes the subtype definition before deleting the subtype declaration itself.

```
~IIR_SubtypeDeclaration();
```

11.9.4 Predefined Public Data Elements

The predefined public data elements consist of a list of attributes which may be associated with the subtype declaration.

```
IIR_AttributeSpecificationList          attributes;
```

11.10 IIR_NatureDeclaration

11.10.1 Derived Class Description

The predefined **IIR_NatureDefinition** class represents the explicit declaration of a new nature definition.

11.10.2 Properties

TABLE 127. IIR_NatureDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_NATURE_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.10.3 Predefined Public Methods

IIR_NatureDeclaration methods other than the constructor must be applied to a valid IIR_NatureDeclaration. All of the following methods are atomic.

11.10.3.1 Constructor Method

The constructor method initializes a nature declaration with an unspecified source location, an unspecified declarator and unspecified nature definition (forward declaration of nature).

```
        IIR_NatureDeclaration();
```

11.10.3.2 Nature Definition Methods

Nature definition methods associate a separately constructed nature definition with the nature declaration

```
    void
        set_nature(IIR_NatureDefinition*      nature);
    IIR_NatureDefinition*
        get_nature();
```

11.10.3.3 Destructor Method

The destructor method deletes the nature definition before deleting the nature declaration itself.

```
~IIR_NatureDeclaration();
```

11.10.4 Predefined Public Data Elements

The predefined public data elements consist of a list of attributes which may be associated with the nature declaration.

```
IIR_AttributeSpecificationList          attributes;
```

11.11 IIR_SubnatureDeclaration

11.11.1 Derived Class Description

The predefined **IIR_SubnatureDeclaration** class represents the explicit declaration of a new subnature definition.

11.11.2 Properties

TABLE 128. IIR_SubnatureDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SUBNATURE_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.11.3 Predefined Public Methods

IIR_SubnatureDeclaration methods other than the constructor must be applied to a valid IIR_SubnatureDeclaration. All of the following methods are atomic.

11.11.3.1 Constructor Method

The constructor method initializes a subnature declaration with an unspecified source location, an unspecified declarator and unspecified nature definition (forward declaration of subnature).

```
    IIR_SubnatureDeclaration();
```

11.11.3.2 Subnature Definition Methods

Subnature definition methods associate a separately constructed subnature definition with the subnature declaration

```
void
    set_subnature (IIR_SubnatureDefinition*      subnature);
IIR_SubnatureDefinition*
    get_subnature();
```

11.11.3.3 Destructor Method

The destructor method deletes the subnature definition before deleting the subnature declaration itself.

```
~IIR_SubnatureDeclaration();
```

11.11.4 Predefined Public Data Elements

The predefined public data elements consist of a list of attributes which may be associated with the subnature declaration.

```
IIR_AttributeSpecificationList          attributes;
```

11.12 IIR_ObjectDeclaration

11.12.1 Derived Class Description

the predefined **IIR_ObjectDeclaration** class represents a constant, variable, signal or file declaration (as derived classes).

11.12.2 Properties

TABLE 129. IIR_Object Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Declaration**
Predefined child classes	**IIR_ConstantDeclaration** **IIR_VariableDeclaration** **IIR_SharedVariableDeclaration** **IIR_SignalDeclaration** **IIR_FileDeclaration** **IIR_TerminalDeclaration** **IIR_QuantityDeclaration**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class

11.12.3 Predefined Public Methods

All IIR_ObjectDeclarations methods must be applied to a valid IIR_ObjectDeclaration class and are atomic.

11.12.3.1 Subtype Definition Methods

Subtype definition methods associate a separately constructed subtype definition with the declaration

```
void
    set_subtype(IIR_TypeDefinition*      subtype);
IIR_TypeDefinition*
    get_subtype();
```

11.12.4 Predefined Public Data Elements

The predefined public data elements consist of a list of attributes which may be associated with the object declaration.

```
IIR_AttributeSpecificationList          attributes;
```

11.13 IIR_ConstantDeclaration

11.13.1 Derived Class Description

The predefined **IIR_ConstantDeclaration** class represent named values which may be assigned exactly once during elaboration. Implementations may update a constant value, such as in the case of a loop iterator, however no programmed assignments may occur to a constant declaration.

11.13.2 Properties

TABLE 130. IIR_ConstantDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONSTANT_DECLARATION**
Parent class	**IIR_ObjectDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.13.3 Predefined Public Methods

IIR_ConstantDeclaration methods other than the constructor must be applied to a valid IIR_ConstantDeclaration. All of the following methods are atomic.

11.13.3.1 Constructor Method

The constructor method initializes the constant declaration with an unspecified source location, unspecified declarator, unspecified subtype and default initial value.

```
    IIR_ConstantDeclaration(),
```

11.13.3.2 Value Methods

Value methods refer to a declaration’s value at the time the call is made.

```
void
    set_value(     IIR*                    value);
IIR*
    get_value();
```

11.13.3.3 Destructor Method

The destructor method deletes each of the initializer value (if present) followed by the declaration itself.

```
~IIR_ConstantDeclaration();
```

11.14 IIR_VariableDeclaration

11.14.1 Derived Class Description

The predefined **IIR_VariableDeclaration** class represents variables which may take on a sequence of values as execution proceeds.

11.14.2 Properties

TABLE 131. IIR_VariableDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_VARIABLE_DECLARATION**
Parent class	**IIR_ObjectDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.14.3 Predefined Public Methods

IIR_VariableDeclaration methods other than the constructor must be applied to a valid IIR_VariableDeclaration. All of the following methods are atomic.

11.14.3.1 Constructor Method

The constructor method initializes a variable declaration an unspecified source location, an unspecified declarator, an unspecified subtype and default initial value.

```
    IIR_VariableDeclaration(),
```

11.14.3.2 Value Methods

Value methods refer to a declaration's value. Before execution, this is the initial value. During execution this is the value at the time the call is made.

```
void
    set_value(IIR*        value):
IIR*
    get_value();
```

11.14.3.3 Destructor Method

The destructor method deletes each of the initializer value (if present) followed by the declaration itself.

```
~IIR_VariableDeclaration();
```

11.15 IIR_SharedVariableDeclaration

11.15.1 Derived Class Description

The predefined **IIR_SharedVariableDeclaration** class represents variables which may take on a sequence of values, assigned from more than one execution thread.

11.15.2 Properties

TABLE 132. IIR_SharedVariableDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SHARED_VARIABLE_DECLARATION**
Parent class	**IIR_ObjectDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.15.3 Predefined Public Methods

IIR_SharedVariableDeclaration methods other than the constructor must be applied to a valid IIR_SharedVariableDeclaration. All of the following methods are atomic.

11.15.3.1 Constructor Method

The constructor method initializes a shared variable declaration an unspecified source location, an unspecified declarator, an unspecified subtype and default initial value.

```
IIR_VariableDeclaration();
```

11.15.3.2 Value Methods

Value methods refer to a declaration's value. Before execution, this is the initial value. During execution this is the value at the time the call is made.

```
void
    set_value(IIR*          value):
IIR*
    get_value();
```

11.15.3.3 Destructor Method

The destructor method deletes each of the initializer value (if present) followed by the shared variable declaration itself.

```
~IIR_SharedVariableDeclaration();
```

11.16 IIR_SignalDeclaration

11.16.1 Derived Class Description

The predefined **IIR_VariableDeclaration** class represents signal declarations which may take on a sequence of values as execution proceeds.

11.16.2 Properties

TABLE 133. IIR_SignalDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SIGNAL_DECLARATION**
Parent class	**IIR_ObjectDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.16.3 Predefined Public Methods

IIR_SignalDeclaration methods other than the constructor must be applied to a valid IIR_SignalDeclaration. All of the following methods are atomic.

11.16.3.1 Constructor Method

The constructor method initializes a signal declaration with an unspecified source location, an unspecified declarator, an unspecified subtype, neither bus nor register signal kind and default initial value.

```
    IIR_SignalDeclaration();
```

11.16.3.2 Value Methods

Value methods refer to a declaration's value. Before execution, this is the initial value. During execution this is the value of the driver at the time the call is made.

```
void
    set_value(IIR*         value):
IIR*
    get_value();
```

11.16.3.3 Signal Kind Methods

.

```
void
    set_signal_kind(IR_SignalKind         signal_kind);
IR_SignalKind
    get_signal_kind();
```

11.16.3.4 Destructor Method

The destructor method deletes each of the initializer value (if present) followed by the declaration itself.

```
    ~IIR_SignalDeclaration();
```

11.17 IIR_FileDeclaration

11.17.1 Derived Class Description

The predefined **IIR_FileDeclaration** class represents an instance of a specific external file.

11.17.2 Properties

TABLE 134. IIR_FileDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_FILE_DECLARATION**
Parent class	**IIR_ObjectDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.17.3 Predefined Public Methods

IIR_FileDeclaration methods other than the constructor must be applied to a valid IIR_FileDeclaration. All of the following methods are atomic.

11.17.3.1 Constructor Method

The constructor method initializes a value file declaration with an unspecified source location, an unspecified declarator, an unspecified subtype, an unspecified file open expression, and an unspecified file logical name.

```
IIR_FileDeclaration();
```

11.17.3.2 File Open Expression Methods

This method refers to the modes with which a file declaration is opened.

```
void
    set_file_open_expression(IIR*        file_open_expression);
IIR*
    get_file_open_expression();
```

11.17.3.3 File Logical Name Methods

File Logical names refer to the name of the file which is opened.

```
void
    set_file_logical_name(IIR*            file_open_expression);
IIR*
    get_file_logical_name();
```

11.17.3.4 Destructor Method

The destructor method deletes the file open expression and file logical name before deleting the file declaration object itself.

```
    ~IIR_FileDeclaration();
```

11.18 IIR_TerminalDeclaration

11.18.1 Derived Class Description

The predefined **IIR_TerminalDeclaration** class.

11.18.2 Properties

TABLE 135. **IIR_TerminalDeclaration Properties**

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_TERMINAL_DECLARATION**
Parent class	**IIR_ObjectDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.18.3 Predefined Public Methods

IIR_TerminalDeclaration methods other than the constructor must be applied to a valid IIR_TerminalDeclaration. All of the following methods are atomic.

11.18.3.1 Constructor Method

The constructor method initializes a terminal declaration with an unspecified source location, an unspecified declarator and an unspecified nature.

```
IIR_TerminalDeclaration();
```

11.18.3.2 Nature Methods

```
void
    set_nature(IIR_NatureDefinition*      nature):
IIR_NatureDefinition*
    get_nature();
```

11.18.3.3 Destructor Method

```
~IIR_TerminalDeclaration();
```

11.19 IIR_QuantityDeclaration

11.19.1 Derived Class Description

The predefined **IIR_QuantityDeclaration** class.

11.19.2 Properties

TABLE 136. **IIR_QuantityDeclaration Properties**

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_QUANTITY_DECLARATION**
Parent class	**IIR_ObjectDeclaration**
Predefined child classes	**IIR_FreeQuantityDeclaration** **IIR_BranchQuantityDeclaration**
Instantiation?	Indirectly via child classes
Application-specific data elements	Via extension class
Public data elements	None

11.19.3 Predefined Public Methods

IIR_QuantityDeclaration methods other than the constructor must be applied to a valid IIR_QuantityDeclaration. All of the following methods are atomic.

11.20 IIR_FreeQuantityDeclaration

11.20.1 Derived Class Description

The predefined **IIR_FreeQuantityDeclaration** class.

11.20.2 Properties

TABLE 137. IIR_FreeQuantityDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_FREE_QUANTITY_DECLARATION**
Parent class	**IIR_QuantityDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.20.3 Predefined Public Methods

IIR_FreeQuantityDeclaration methods other than the constructor must be applied to a valid IIR_FreeQuantityDeclaration. All of the following methods are atomic.

11.20.3.1 Constructor Method

The constructor method initializes a terminal declaration with an unspecified source location, an unspecified declarator and an unspecified nature.

```
IIR_FreeQuantityDeclaration();
```

11.20.3.2 Nature Methods

```
void
    set_subnature_indication(IIR_NatureDefinition*      value):
IIR_NatureDefinition*
    get_subnature_indication();
```

11.20.3.3 Value Methods

```
void
    set_value(IIR*          value):
IIR*
    get_value();
```

11.20.3.4 Destructor Method

```
    ~IIR_FreeQuantityDeclaration();
```

11.21 IIR_AcrossQuantityDeclaration

11.21.1 Derived Class Description

The predefined **IIR_AcrossQuantityDeclaration** class.

11.21.2 Properties

TABLE 138. IIR_AcrossQuantityDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_ACROSS_QUANTITY_DECLARATION**
Parent class	**IIR_QuantityDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.21.3 Predefined Public Methods

IIR_AcrossQuantityDeclaration methods other than the constructor must be applied to a valid IIR_AcrossQuantityDeclaration. All of the following methods are atomic.

11.21.3.1 Constructor Method

The constructor method initializes a across quantity declaration with an unspecified source location, an unspecified declarator and an unspecified nature.

```
    IIR_AcrossQuantityDeclaration();
```

11.21.3.2 Aspect Methods

```
void
    set_expression(IIR*                    expression);
IIR*
    get_expression();

void
    set_tolerance(IIR*                     tolerance);
IIR*
    get_tolerance();
```

11.21.3.3 Plus Terminal Name Methods

```
void
    set_plus_terminal_name(IIR*          plus_terminal_name);
IIR*
    get_plus_terminal_name();
```

11.21.3.4 Minus Terminal Name Methods

```
void
    set_minus_terminal_name(IIR*         minus_terminal_name);
IIR*
    get_minus_terminal_name();
```

11.21.3.5 Destructor Method

```
    ~IIR_AcrossQuantityDeclaration();
```

11.21.4 Predefined Public Data Elements

There are no predefined public data elements

11.22 IIR_NoiseSourceQuantityDeclaration

11.22.1 Derived Class Description

The predefined **IIR_NoiseSourceQuantityDeclaration** class.

11.22.2 Properties

TABLE 139. IIR_NoiseSourceQuantityDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_NOISE_SOURCE_QUANTITY_DECLARATION**
Parent class	**IIR_QuantityDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.22.3 Predefined Public Methods

IIR_NoiseSourceQuantityDeclaration methods other than the constructor must be applied to a valid IIR_NoiseSourceQuantityDeclaration. All of the following methods are atomic.

11.22.3.1 Constructor Method

The constructor method initializes a noisesource quantity declaration with an unspecified source location, an unspecified declarator and an unspecified nature.

```
IIR_NoiseSourceQuantityDeclaration();
```

11.22.3.2 Subnature Indication Methods

```
void
    set_subnature_indication(IIR_NatureDefinition*subnature_indication);
IIR_NatureDefinition*
    get_subnature_indication();
```

11.22.3.3 Magnitude Simple Expression Methods

```
void
    set_magnitude_simple_expression(IIR*         value);
IIR*
    get_magnitude_simple_expression();
```

11.22.3.4 Destructor Method

```
    ~IIR_NoiseSourceQuantityDeclaration();
```

11.22.4 Predefined Public Data Elements

There are no predefined public data elements

11.23 IIR_SpectrumSourceQuantityDeclaration

11.23.1 Derived Class Description

The predefined **IIR_SpectrumSourceQuantityDeclaration** class.

11.23.2 Properties

TABLE 140. IIR_SpectrumSourceQuantityDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SPECTRUM_SOURCE_QUANTITY_DECLARATION**
Parent class	**IIR_QuantityDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.23.3 Predefined Public Methods

IIR_SpectrumSourceQuantityDeclaration methods other than the constructor must be applied to a valid IIR_SpectrumSourceQuantityDeclaration. All of the following methods are atomic.

11.23.3.1 Constructor Method

The constructor method initializes a spectrumsource quantity declaration with an unspecified source location, an unspecified declarator and an unspecified nature.

```
    IIR_SpectrumSourceQuantityDeclaration();
```

11.23.3.2 Subnature Indication Methods

```
void
    set_subnature_indication(IIR_NatureDefinition*       subnature_indication);
IIR*
    get_subnature_indication();
```

11.23.3.3 Magnitude Simple Expression Methods

```
void
    set_magnitude_simple_expressionIIR*           value);
IIR*
    get_magnitude_simple_expression();
```

11.23.3.4 Phase Simple Expression Methods

```
void
    set_phase_simple_expression(IIR*            value);
IIR*
    get_phase_simple_expression();
```

11.23.3.5 Destructor Method

```
    ~IIR_SpectrumSourceQuantityDeclaration();
```

11.23.4 Predefined Public Data Elements

There are no predefined public data elements

11.24 IIR_ThroughQuantityDeclaration

11.24.1 Derived Class Description

The predefined **IIR_ThroughQuantityDeclaration** class.

11.24.2 Properties

TABLE 141. IIR_ThroughQuantityDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_THROUGH_QUANTITY_DECLARATION**
Parent class	**IIR_QuantityDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.24.3 Predefined Public Methods

IIR_ThroughQuantityDeclaration methods other than the constructor must be applied to a valid IIR_ThroughQuantityDeclaration. All of the following methods are atomic.

11.24.3.1 Constructor Method

The constructor method initializes a through quantity declaration with an unspecified source location, an unspecified declarator and an unspecified nature.

```
IIR_ThroughQuantityDeclaration();
```

11.24.3.2 Across Aspect Methods

```
void
    set_across_aspect_expression(IIR*     across_aspect_expression);
IIR*
    get_across_aspect_expression();
void
    set_across_aspect_tolerence(IIR*      across_aspect_tolerence);
IIR*
    get_across_aspect_tolerence();
```

11.24.3.3 Aspect Methods

```
void
    set_expression(IIR*                   expression);
IIR*
    get_expression();
void
    set_tolerance(IIR*                    tolerance);
IIR*
    get_tolerance();
```

11.24.3.4 Plus Terminal Name Methods

```
void
    set_plus_terminal_name(IIR*           plus_terminal_name);
IIR*
    get_plus_terminal_name();
```

11.24.3.5 Minus Terminal Name Methods

```
void
    set_minus_terminal_name(IIR*          minus_terminal_name);
IIR*
    get_minus_terminal_name();
```

11.24.3.6 Destructor Method

```
    ~IIR_ThroughQuantityDeclaration();
```

11.24.4 Predefined Public Data Elements

There are no predefined public data elements

11.25 IIR_InterfaceDeclaration

11.25.1 Derived Class Description

The predefined **IIR_InterfaceDeclaration** class refers to constants, variables, signals and files present at an elaborated interface.

11.25.2 Properties

TABLE 142. IIR_InterfaceDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98, VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Declaration**
Predefined child classes	**IIR_ConstantInterfaceDeclaration** **IIR_VariableInterfaceDeclaration** **IIR_SignalInterfaceDeclaration** **IIR_FileInterfaceDeclaration** **IIR_TerminalInterfaceDeclaration** **IIR_QuantityInterfaceDeclaration**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class

11.25.3 Predefined Public Methods

All IIR_InterfaceDeclarations methods must be applied to a valid IIR_InterfaceDeclaration class and are atomic.

11.25.3.1 Mode Methods

Mode methods denote the mode associated with an interface object. The mode must be an IIR_Option from the set IIR_NO_MODE, IIR_IN_MODE, IIR_OUT_MODE, IIR_INOUT_MODE, IIR_BUFFER_MODE, IIR_LINKAGE_MODE.

```
void
    set_mode(IIR_Mode     mode);
IIR_Mode
    get_mode();
```

11.25.3.2 Subtype Definition Methods

Subtype definition methods associate a separately constructed subtype definition with the interface declaration

```
void
    set_subtype(IIR_TypeDefinition*       subtype);
IIR_TypeDefinition*
    get_subtype();
```

11.25.3.3 Value Methods

Value methods refer to a declaration's value at the time the call is made.

```
void
    set_value(IIR*        value);
IIR*
    get_value();
```

11.25.4 Predefined Public Data Elements

The predefined public data elements consist of a list of attributes which may be associated with the interface declaration.

```
IIR_AttributeSpecificationList           attributes;
```

11.26 IIR_ConstantInterfaceDeclaration

11.26.1 Derived Class Description

The predefined **IIR_ConstantInterfaceDeclaration** class represent named values which may be assigned exactly once during elaboration.

11.26.2 Properties

TABLE 143. IIR_ConstantInterfaceDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONSTANT_INTERFACE_DECLARATION**
Parent class	**IIR_InterfaceDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.26.3 Predefined Public Methods

IIR_ConstantInterfaceDeclaration methods other than the constructor must be applied to a valid IIR_ConstantInterfaceDeclaration. All of the following methods are atomic.

11.26.3.1 Constructor Method

The constructor method initializes a constant interface declaration with an unspecified source location, an unspecified declarator, an unspecified subtype and default initial value. The mode of a constant interface declaration is implicitly IIR_IN_OPTION.

```
IIR_ConstantInterfaceDeclaration();
```

11.26.3.2 Destructor Method

The destructor method deletes the initial value (if present) before deleting the constant interface declaration itself.

```
~IIR_ConstantInterfaceDeclaration();
```

11.27 IIR_VariableInterfaceDeclaration

11.27.1 Derived Class Description

The predefined **IIR_VariableInterfaceDeclaration** class represents interface variables which may take on a sequence of values as execution proceeds.

11.27.2 Properties

TABLE 144. IIR_VariableInterfaceDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_VARIABLE_INTERFACE_DECLARATION**
Parent class	**IIR_InterfaceDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.27.3 Predefined Public Methods

IIR_VariableInterfaceDeclaration methods other than the constructor must be applied to a valid IIR_VariableInterfaceDeclaration. All of the following methods are atomic.

11.27.3.1 Constructor Method

The constructor method initializes a variable interface declaration with an unspecified source code location, an unspecified declarator, an unspecified mode, an unspecified subtype, and default initial value.

```
IIR_VariableInterfaceDeclaration();
```

11.27.3.2 Destructor Method

The destructor method deletes the initial value, if present, before deleting the variable interface declaration itself.

```
~IIR_VariableInterfaceDeclaration();
```

11.28 IIR_SignalInterfaceDeclaration

11.28.1 Derived Class Description

The predefined **IIR_SignalInterfaceDeclaration** class represents signal interface declarations which may take on a sequence of values as execution proceeds.

11.28.2 Properties

TABLE 145. IIR_SignalInterfaceDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SIGNAL_INTERFACE_DECLARATION**
Parent class	**IIR_InterfaceDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.28.3 Predefined Public Methods

IIR_SignalInterfaceDeclaration methods other than the constructor must be applied to a valid IIR_SignalInterfaceDeclaration. All of the following methods are atomic.

11.28.3.1 Constructor Method

The constructor method initializes a signal interface declaration with an unspecified source location, an unspecified declarator, an unspecified mode, an unspecified subtype, an unspecified signal kind and default initial value.

```
        IIR_SignalInterfaceDeclaration();
```

11.28.3.2 Signal Kind Methods

Signal kind methods denote a signal interface declaration having the signal kinds IIR_NO_KIND, IIR_BUS_KIND or IIR_REGISTER_KIND.

```
    void
        set_signal_kind(IR_SignalKind         signal_kind);
    IR_SignalKind
        get_signal_kind();
```

11.28.3.3 Destructor Method

The destructor method deletes the initial value (if present) before deleting the signal interface declaration.

```
~IIR_SignalInterfaceDeclaration();
```

11.29 IIR_FileInterfaceDeclaration

11.29.1 Derived Class Description

The predefined **IIR_FileInterfaceDeclaration** class represents an instance of a specific external file.

11.29.2 Properties

TABLE 146. IIR_FileInterfaceDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_FILE_INTERFACE_DECLARATION**
Parent class	**IIR_InterfaceDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.29.3 Predefined Public Methods

IIR_FileInterfaceDeclaration methods other than the constructor must be applied to a valid IIR_FileInterfaceDeclaration. All of the following methods are atomic.

11.29.3.1 Constructor Method

The constructor method initializes a file interface declaration object with an unspecified declarator and an unspecified subtype.

```
IIR_FileInterfaceDeclaration();
```

11.29.3.2 Destructor Method

The destructor method deletes the file interface declaration itself.

```
~IIR_FileInterfaceDeclaration();
```

11.30 IIR_TerminalInterfaceDeclaration

11.30.1 Derived Class Description

The predefined **IIR_TerminalInterfaceDeclaration** class.

11.30.2 Properties

TABLE 147. IIR_TerminalInterfaceDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_TERMINAL_INTERFACE_DECLARATION**
Parent class	**IIR_InterfaceDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.30.3 Predefined Public Methods

IIR_TerminalInterfaceDeclaration methods other than the constructor must be applied to a valid IIR_TerminalInterfaceDeclaration. All of the following methods are atomic.

11.30.3.1 Constructor Method

The constructor method initializes a terminal interface declaration with an unspecified source location, an unspecified declarator and an unspecified nature.

```
IIR_TerminalInterfaceDeclaration();
```

11.30.3.2 Subnature Indiciation Methods

```
void
    set_subnature_indication(IIR_NatureDefinition*      subnature_indication):
IIR_NatureDefinition*
    get_subnature_indication();
```

11.30.3.3 Destructor Method

```
~IIR_TerminalInterfaceDeclaration();
```

11.31 IIR_QuantityInterfaceDeclaration

11.31.1 Derived Class Description

The predefined **IIR_QuantityInterfaceDeclaration** class.

11.31.2 Properties

TABLE 148. IIR_QuantityInterfaceDeclaration Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_QUANTITY_INTERFACE_DECLARATION**
Parent class	**IIR_InterfaceDeclaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.31.3 Predefined Public Methods

IIR_QuantityInterfaceDeclaration methods other than the constructor must be applied to a valid IIR_QuantityInterfaceDeclaration. All of the following methods are atomic.

11.31.3.1 Constructor Method

```
IIR_QuantityInterfaceDeclaration();
```

11.31.3.2 Subnature Indiciation Methods

```
void
    set_subnature_indication(IIR_NatureDefinition*      subnature_indication):
IIR_NatureDefinition*
    get_subnature_indication();
```

11.31.3.3 Destructor Method

```
~IIR_QuantityInterfaceDeclaration();
```

11.32 IIR_AliasDeclaration

11.32.1 Derived Class Description

The predefined **IIR_AliasDeclaration** class represent alternative names for a pre-existing entity.

11.32.2 Properties

TABLE 149. **IIR_AliasDeclaration Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_ALIAS_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.32.3 Predefined Public Methods

IIR_AliasDeclaration methods other than the constructor must be applied to a valid IIR_AliasDeclaration. All of the following methods are atomic.

11.32.3.1 Constructor Method

The constructor method initializes an alias declaration with an unspecified declarator, an unspecified subtype and an unspecified name.

```
IIR_AliasDeclaration();
```

11.32.3.2 Subtype Methods

Subtype Methods refer to the alias's subtype.

```
void
    set_subtype(IIR_TypeDefinition*       subtype);
IIR_TypeDefinition*
    get_subtype();
```

11.32.3.3 Name Methods

Name Methods refer to the existing entity named by the alias.

```
void
    set_name(IIR*          name);
IIR*
    get_name();
```

11.32.3.4 Destructor Method

The destructor method deletes the alias declaration itself.

```
    ~IIR_AliasDeclaration();
```

11.33 IIR_AttributeDeclaration

11.33.1 Derived Class Description

The predefined **IIR_AttributeDeclaration** class represent a named and typed attribute which may be associated with existing entities through an attribute specification.

11.33.2 Properties

TABLE 150. IIR_AttributeDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_ATTRIBUTE_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.33.3 Predefined Public Methods

IIR_AttributeDeclaration methods other than the constructor must be applied to a valid IIR_AttributeDeclaration. All of the following methods are atomic.

11.33.3.1 Constructor Method

The constructor method initializes an attribute specification using an unspecified source location, an unspecified declarator and an unspecified type mark.

```
IIR_AttributeDeclaration();
```

11.33.3.2 Subtype Methods

Subtype Methods refer to the attribute declaration's subtype.

```
void
    set_subtype(IIR_TypeDefinition*       subtype);
IIR_TypeDefinition*
    get_subtype();
```

11.33.3.3 Destructor Method

The destructor method deletes the attribute subtype itself.

```
~IIR_AttributeDeclaration();
```

11.34 IIR_ComponentDeclaration

11.34.1 Derived Class Description

The predefined **IIR_ComponentDeclaration** class represents a place-holder for an entity interface (allowing delayed binding of a component instance to a specific entity/architecture pair).

11.34.2 Properties

TABLE 151. IIR_ComponentDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_COMPONENT_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.34.3 Predefined Public Methods

IIR_ComponentDeclaration methods other than the constructor must be applied to a valid IIR_ComponentDeclaration. All of the following methods are atomic.

11.34.3.1 Constructor Method

The constructor method initializes a component declaration with an unspecified source location, an unspecified declarator, no generic interface declarations, no signal interface declarations, and no attribute specifications.

```
IIR_ComponentDeclaration();
```

11.34.3.2 Destructor Method

The destructor method deletes an generics, ports and attributes before deleting the component declaration itself.

```
~IIR_ComponentDeclaration();
```

11.34.4 Predefined Public Data

The component declaration includes three predefined public data elements:

```
IIR_GenericList                    local_generic_clause;
IIR_PortList                       local_port_clause;
IIR_AttributeSpecificationList     attributes;
```

11.35 IIR_GroupDeclaration

11.35.1 Derived Class Description

The predefined **IIR_GroupDeclaration** class represents explicit, named collections of entities corresponding to a group template declaration.

11.35.2 Properties

TABLE 152. **IIR_GroupDeclaration Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_GROUP_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.35.3 Predefined Public Methods

IIR_GroupDeclaration methods other than the constructor must be applied to a valid IIR_GroupDeclaration. All of the following methods are atomic.

11.35.3.1 Constructor Method

The constructor method initializes a group declaration with an unspecified source location, an unspecified declarator, an unspecified group template and no group constituents.

```
IIR_GroupDeclaration();
```

11.35.3.2 Group Template Methods

These methods refer to a pre-existing group template from which this group declaration is built.

```
void
    set_group_template(IIR_Name*          group_template_name);
IIR_Name*
    get_group_template_name();
```

11.35.3.3 Destructor Method

The destructor method deletes each of the group constituents and attributes before deleting the group declaration itself.

```
~IIR_GroupDeclaration();
```

11.35.4 Predefined Public Data

```
IIR_DesignatorList                  group_constituent_list;
IIR_AttributeSpecificationList      attributes;
```

11.36 IIR_GroupTemplateDeclaration

11.36.1 Derived Class Description

The predefined **IIR_GroupTemplateDeclaration** class declares a sequence of entity class entries from which group declarations may be made.

11.36.2 Properties

TABLE 153. IIR_GroupTemplateDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_GROUP_TEMPLATE_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.36.3 Predefined Public Methods

IIR_GroupTemplateDeclaration methods other than the constructor must be applied to a valid IIR_GroupTemplateDeclaration. All of the following methods are atomic.

11.36.3.1 Constructor Method

The constructor method initializes the group template declaration with an unspecified source location and an unspecified declarator with no predefined entity class entries.

```
IIR_GroupTemplateDeclaration();
```

11.36.3.2 Destructor Method

The destructor method deletes each of the entity class entries and attributes before deleting the group template object itself.

```
~IIR_GroupTemplateDeclaration();
```

11.36.4 Predefined Public Data

```
IIR_EntityClassEntryList          entity_class_entry_list;
```

11.37 IIR_LibraryDeclaration

Derived Class Description

The predefined **IIR_LibraryUnit** class represents a named set of zero or more library units.

11.37.1 Properties

TABLE 154. IIR_LibraryDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_LIBRARY_DECLARATION**
Parent class	**IIR_Declaration**
Predefined child classes	**none**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class

11.37.2 Predefined Public Methods

All IIR_LibraryDeclaration methods, except for the constructor, must be applied to a valid IIR_LibraryDeclaration class and are atomic.

11.37.3 Predefined Public Data Elements

All library units have two predefined public data elements:

```
IIR_LibraryUnitList          primary_units;
```

11.38 IIR_LibraryUnit

11.38.1 Derived Class Description

The predefined **IIR_LibraryUnit** class is a parent representing entity, architecture, package, package body and configuration declarations.

11.38.2 Properties

TABLE 155. IIR_LibraryUnit Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_LibraryUnit**
Predefined child classes	**IIR_EntityDeclaration** **IIR_ArchitectureDeclaration** **IIR_PackageDeclaration** **IIR_PackageBodyDeclaration** **IIR_ConfigurationDeclaration**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class

11.38.3 Predefined Public Methods

All IIR_LibraryUnit methods must be applied to a valid IIR_LibraryUnit class and are atomic.

11.38.4 Predefined Public Data Elements

All library units have two predefined public data elements:

```
IIR_DeclarationList              context_items;
IIR_AttributeSpecificationList   attributes;
```

11.39 IIR_EntityDeclaration

11.39.1 Derived Class Description

The predefined **IIR_EntityDeclaration** class represents VHDL entities.

11.39.2 Properties

TABLE 156. IIR_EntityDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_ENTITY_DECLARATION**
Parent class	**IIR_LibraryUnit**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.39.3 Predefined Public Methods

IIR_EntityDeclaration methods other than the constructor must be applied to a valid IIR_EntityDeclaration. All of the following methods are atomic.

11.39.3.1 Constructor Method

The constructor method initializes an entity declaration using an unspecified source location, an unspecified declarator, no generics, no ports, no entity declarations, no entity statements and no attributes.

```
IIR_EntityDeclaration();
```

11.39.3.2 Last Architecture Methods

```
void
    set_last_analyzed_architecture(IIR_ArchitectureDeclaration*  architecture);
IIR_ArchitectureDeclaration*
    get_last_analyzed_architecture();
```

11.39.3.3 Destructor Method

The destructor method deletes any generics, ports, entity declarations, entity statements and attributes before deleting the entity declaration itself.

```
~IIR_EntityDeclaration();
```

11.39.4 Predefined Public Data Elements

Entity declarations include five predefined public data elements:

```
IIR_GenericList                  generic_clause;
IIR_PortList                     port_clause;
IIR_DeclarationList              entity_declarative_part;
IIR_StatementList                entity_statement_part;
IIR_LibraryUnitList              architectures;
```

11.40 IIR_ArchitectureDeclaration

11.40.1 Derived Class Description

The predefined **IIR_ArchitectureDeclaration** class represents one of potentially several implementations of an entity.

11.40.2 Properties

TABLE 157. **IIR_ArchitectureDeclaration Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_ARCHITECTURE_DECLARATION**
Parent class	**IIR_LibraryUnit**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.40.3 Predefined Public Methods

IIR_ArchitectureDeclaration methods other than the constructor must be applied to a valid IIR_ArchitectureDeclaration. All of the following methods are atomic.

11.40.3.1 Constructor Method

The constructor method initializes an architecture declaration using an unspecified source location, an unspecified architecture declarator, an unspecified entity name, no architecture declarations, no architecture statements and no attributes.

```
    IIR_ArchitectureDeclaration();
```

11.40.3.2 Entity Methods

The entity methods identify the design entity with which this architecture is associated. The entity is established when the architecture is constructed and may not change.

```
void
    set_entity(          IIR_EntityDeclaration          entity);

IIR_EntityDeclaration*
    get_entity();
```

11.40.3.3 Destructor Method

The destructor method deletes all architecture declarations, architecture statements and attributes, disassociates the architecture from it's entity, then deletes the architecture declaration itself.

```
~IIR_ArchitectureDeclaration();
```

11.40.4 Predefined Public Data

```
IIR_DeclarationList              architectecture_declarative_part;
IIR_StatementList                architecture_statement_part;
```

11.41 IIR_PackageDeclaration

11.41.1 Derived Class Description

The predefined **IIR_PackageDeclaration** class represents collections of declarations which are elaborated at most once, as a collection.

11.41.2 Properties

TABLE 158. IIR_PackageDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IIR_PACKAGE_DECLARATION**
Parent class	**IIR_LibraryUnit**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.41.3 Predefined Public Methods

IIR_PackageDeclaration methods other than the constructor must be applied to a valid IIR_PackageDeclaration. All of the following methods are atomic.

11.41.3.1 Constructor Method

The constructor method initializes a package declaration using an unspecified source location, an unspecified declarator, no package declarations, no attributes and no package body.

```
IIR_PackageDeclaration();
```

11.41.3.2 Package Body Method

This method refers to the package's body, if one exists. The package body is defined as part of constructing the package body object and may not be altered.

```
IIR_PackageBodyDeclaration*
    get_package_body();
```

11.41.3.3 Destructor Method

The destructor method deletes the package body (if one exists), the package declarations, any package attributes, and finally the package declaration object itself.

```
~IIR_PackageDeclaration();
```

11.41.4 Predefined Public Data Element

Packages include one predefined public data element, a list of declarations appearing in the package:

```
IIR_DeclarationList                package_declarative_part;
```

11.42 IIR_PackageBodyDeclaration

11.42.1 Derived Class Description

The predefined **IIR_PackageBodyDeclaration** class represents the optional implementation part of a package declaration.

11.42.2 Properties

TABLE 159. IIR_PackageBodyDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_PACKAGE_BODY_DECLARATION**
Parent class	**IIR_LibraryUnit**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.42.3 Predefined Public Methods

IIR_PackageBodyDeclaration methods other than the constructor must be applied to a valid IIR_PackageBodyDeclaration. All of the following methods are atomic.

11.42.3.1 Constructor Method

The constructor method initializes a package body declaration from an unspecified source location and unspecified declarator.

```
IIR_PackageBodyDeclaration();
```

11.42.3.2 Associate Method

The associate method pairs a named package body declaration up with the corresponding package declaration.

```
void
    associate();
```

11.42.3.3 Destructor Method

The destructor method deletes each of the package body declarations (if present) before deleting the package body declaration object itself.

```
~IIR_PackageBodyDeclaration();
```

11.42.4 Predefined Public Data Elements

The package body declaration includes a single predefined public data element:

```
IIR_DeclarationList              package_body_declarative_part;
```

11.43 IIR_ConfigurationDeclaration

11.43.1 Derived Class Description

The predefined **IIR_ConfigurationDeclaration** class represents the delayed binding components to entity/architecture pairs.

11.43.2 Properties

TABLE 160. IIR_ConfigurationDeclaration Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_CONFIGURATION_DECLARATION**
Parent class	**IIR_LibraryUnit**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.43.3 Predefined Public Methods

IIR_ConfigurationDeclaration methods other than the constructor must be applied to a valid IIR_ConfiguraitonDeclaration. All of the following methods are atomic.

11.43.3.1 Constructor Method

The constructor method initializes a configuration declaration with an unspecified source location, an unspecified declarator, an unspecified entity name, and no configuration declarations.

```
        IIR_ConfigurationDeclaration();
```

11.43.3.2 Block Configuration Methods

These methods refer to the block configuration immediately enclosed within a configuration declaration (which must refer to an architecture).

```
    void
        set_block_configuration(      IIR_BlockConfiguration*       block_configuration);
    IIR_BlockConfiguration*
        get_block_configuration();
```

11.43.3.3 Entity Methods

The entity methods identify the design entity with which this configuration is associated. The entity is established when the configuration is constructed and may not change.

```
void
    set_entity(    IIR_EntityDeclaration*                    entity);
IIR_EntityDeclaration*
    get_entity();
```

11.43.3.4 Destructor Method

The destructor method deletes any configuration declarations, the immediately enclosed block configuration, any attribute specifications and finally the configuration declaration itself.

```
    ~IIR_ConfigurationDeclaration();
```

11.43.4 Predefined Public Data Elements

```
IIR_DeclarationList       configuration_declarative_part;
```

11.44 IIR_PhysicalUnit

11.44.1 Derived Class Description

The predefined **IIR_PhysicalUnit** class represents physical units within a list of such physical units (and indirectly within a physical type or subtype definition).

11.44.2 Properties

TABLE 161. IIR_PhysicalUnit Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_PHYSICAL_UNIT**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.44.3 Predefined Public Methods

IIR_PhysicalUnit methods other than the constructor must be applied to a valid IIR_PhysicalUnit. All of the following methods are atomic.

11.44.3.1 Constructor Method

The constructor method initializes a physical unit with an unspecified source location, an unspecified declarator and an unspecified unit multiplier with no attributes.

```
IIR_PhysicalUnit();
```

11.44.3.2 Multiplier Methods

Multiplier methods refer to the multiplier associated with a physical unit relative to a physical unit's unit name.

```
void
    set_multiplier(IIR*   multiplier);
IIR*
    get_multiplier();
```

11.44.3.3 Unit Name Methods

The unit name identifies an optional physical unit from which the multiplier is measured. Primary units have a unit_name pointing to this physical unit.

```
void
    set_unit_name(IIR_PhysicalUnit*        unit_name);
IIR_PhysicalUnit*
    get_unit_name();
```

11.44.3.4 Destructor Method

The destructor method deletes the multiplier and attributes associated with the physical unit, then the physical unit object itself.

```
    ~IIR_PhysicalUnit();
```

11.44.4 Predefined Public Data Elements

```
IIR_AttributeSpecificationList          attributes;
```

11.45 IIR_AttributeSpecification

11.45.1 Derived Class Description

The predefined **IIR_AttributeSpecification** class decorates a named entity with a previously declared, named attribute and value.

11.45.2 Properties

TABLE 162. IIR_AttributeSpecification Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IIR_ATTRIBUTE_SPECIFICATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.45.3 Predefined Public Methods

IIR_AttributeSpecification methods other than the constructor must be applied to a valid IIR_AttributeSpecification. All of the following methods are atomic.

11.45.3.1 Constructor Method

The constructor method initializes an attribute specification with an unspecified source location, an unspecified attribute designator, an unspecified value and no entity specification.

```
IIR_AttributeSpecification();
```

11.45.3.2 Value Methods

Value methods refer to the value associated with a specific entity and attribute name by the attribute specification.

```
void
    set_value(IIR*        value);
IIR*
    get_value();
```

11.45.3.3 Entity Class Methods

Entity class methods define the entity class associated with the entity name list.

```
void
    set_entity_class(IIR_Identifier*      entity_class);
IIR_Identifier*
    get_entity_class();
```

11.45.3.4 Destructor Method

The destructor method deletes the attribute value and entity specification before deleting the attribute specification itself.

```
    ~IIR_AttributeSpecification();
```

11.45.4 Predefined Public Data Elements

```
IIR_DesignatorList          entity_name_list;
```

11.46 IIR_ConfigurationSpecification

11.46.1 Derived Class Description

The predefined **IIR_ConfigurationSpecification** class specifies binding information associated with IIR_ComponentInstantiationStatements.

11.46.2 Properties

TABLE 163. IIR_ConfigurationSpecification Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_CONFIGURATION_SPECIFICATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.46.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConfigurationSpecification object. All of the following methods are atomic.

11.46.3.1 Constructor Method

The constructor initializes a configuration specification object with an unspecified source location, unspecified component specification and an undefined entity aspect, no generic map elements and no port map elements.:

```
    IIR_ConfigurationSpecification();
```

11.46.3.2 Component Name Method

The component name denotes the component declaration to which all instances in the instantiation list apply.

```
void
    set_component_name(IIR*        component_name);
IIR*
    get_component_name();
```

11.46.3.3 Entity Aspect Method

The entity aspect refers to the design entity (if any) to be associated with this component.

```
void
    set_entity_aspect(IIR_LibraryUnit*    entity_aspect);
IIR_LibraryUnit*
    get_entity_aspect();
```

11.46.3.4 Destructor Method

The configuration specification destructor deletes the component specification and binding indication before deleting the configuration specification object itself.

```
    ~IIR_ConfigurationSpecification();
```

11.46.4 Predefined Public Elements

The configuration specification has three predefined public data elements:

```
IIR_DesignatorList         instantiation_list;
IIR_AssociationList        generic_map_aspect;
IIR_AssociationList        port_map_aspect;
```

11.47 IIR_DisconnectionSpecification

11.47.1 Derived Class Description

The predefined **IIR_DisconnectionSpecification** class denotes the time delay associated with disconnection of a signal's drivers.

11.47.2 Properties

TABLE 164. **IIR_DisconnectionSpecification Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IIR_DISCONNECTION_SPECIFICATION**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.47.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_DisconnectionSpecification object. All of the following methods are atomic.

11.47.3.1 Constructor Method

The constructor initializes a disconnect specification using an unspecified source location, an unspecified signal name, an unspecified type mark, and an unspecified delay.

```
IIR_DisconnectionSpecification();
```

11.47.3.2 Type Mark Methods

The type mark methods specify the guarded signal's type mark.

```
void
    set_type_mark(          IIR_TypeDefinition*          type_mark);
IIR_TypeDefinition*
    get_type_mark();
```

11.47.3.3 Time Expressions Methods

The delay methods specify the delay value associated with the disconnect specification.

```
void
    set_time_expression(          IIR*          time_expression);
IIR*
    get_time_expression();
```

11.47.3.4 Destructor Method

The destructor method deletes the signal name and delay before deleting the disconnect specification itself.

```
    ~IIR_DisconnectionSpecification();
```

11.47.4 Predefined Public Data Elements

```
IIR_DesignatorList                guarded_signal_list;
```

11.48 IIR_Label

11.48.1 Derived Class Description

The predefined **IIR_Label** class associates a label with a sequential statement (concurrent statements have intrinsic labels).

11.48.2 Properties

TABLE 165. IIR_Label Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_LABEL**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

11.48.3 Predefined Public Methods

IIR_Label methods other than the constructor must be applied to a valid IIR_Label. All of the following methods are atomic.

11.48.3.1 Constructor Method

The constructor method initializes a label using an unspecified source location, an unspecified logical name, an unspecified statement and no attributes.

```
IIR_Label();
```

11.48.3.2 Statement Methods

The statement methods refer to the sequential statement being labeled.

```
void
    set_statement(IIR_SequentialStatement*       statement);
IIR_Statement*
    get_statement();
```

11.48.3.3 Destructor Method

The destructor method deletes any attribute specifications before deleting the label object itself.

```
~IIR_Label();
```

11.48.4 Predefined Public Data Elements

```
IIR_AttributeSpecificationList          attributes;
```

11.49 IIR_LibraryClause

11.49.1 Derived Class Description

The predefined **IIR_LibraryClause** class brings an externally defined design library name into direct visibility.

11.49.2 Properties

TABLE 166. IIR_LibraryClause Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_LIBRARY_CLAUSE**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.49.3 Predefined Public Methods

IIR_LibraryClause methods other than the constructor must be applied to a valid IIR_LibraryClause. All of the following methods are atomic.

11.49.3.1 Constructor Method

The constructor method initializes a library clause with an unspecified logical name.

```
IIR_LibraryClause();
```

11.49.3.2 Logical Name Methods

Logical name methods refer to a specific, pre-existing library.

```
void
    set_logical_name(IIR_LibraryDeclaration*     logical_name);
IIR_LibraryDeclaration*
    get_logical_name();
```

11.49.3.3 Destructor Method

The destructor method deletes the library clause itself.

```
~IIR_LibraryClause();
```

11.50 IIR_UseClause

11.50.1 Derived Class Description

The predefined **IIR_UseClause** class brings an existing declaration into direct visibility.

11.50.2 Properties

TABLE 167. IIR_UseClause Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_USE_CLAUSE**
Parent class	**IIR_Declaration**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

11.50.3 Predefined Public Methods

IIR_UseClause methods other than the constructor must be applied to a valid IIR_UseClause. All of the following methods are atomic.

11.50.3.1 Constructor Method

The constructor method initializes a use clause with an unspecified selected name.

```
IIR_UseClause();
```

11.50.3.2 Selected Name Methods

Selected name methods refer to the object being brought into directly visibility.

```
void
    set_selected_name(IIR_Name*           selected_name);
IIR_Name*
    get_selected_name();
```

11.50.3.3 Destructor Method

The destructor method deletes the use clause object itself.

```
~IIR_UseClause();
```

CHAPTER 12

IIR Name Derived Class

Names generally refer to an explicitly or implicitly declared entity. The variety of name forms derived from the IIR_Name class is shown in Table 168 on page 331.

As represented in the IIR information model, a name may always be replaced by its referent. For example the name for a declaration may be replaced by the declaration itself. Translating from a names to its referent(s) is called lookup. Since the actual organization of declarator information is implementation-dependent, IIR provides a general set of methods for name to referent translation. These lookup functions are static methods of the IIR_Name method in order to limit extent of the externally visible IIR name space (and thus impact on non-IIR code linked with an IIR foundation implementation).

TABLE 168. Class hierarchy derived from IIR_Name class

	Level 3 Derived Classes (only classes derived from IIR_Name)		**Level 4 Derived Classes**		**Level 5 Derived Classes**
E	*IIR_SimpleName*				
E	*IIR_SelectedName*				
E	*IIR_SelectedNameByAll*				
E	*IIR_IndexedName*				
E	*IIR_SliceName*				
E	IIR_Attribute	E	*IIR_UserAttribute*		
		E	*IIR_BaseAttriBute*		
		E	*IIR_LeftAttribute*		
		E	*IIR_RightAttribute*		
		E	*IIR_LowAttribute*		
		E	*IIR_HighAttribute*		
		E	*IIR_AscendingAttribute*		

TABLE 168. **Class hierarchy derived from IIR_Name class**

		E	*IIR_ImageAttribute*		
		E	*IIR_ValueAttribute*		
		E	*IIR_PosAttribute*		
		E	*IIR_ValAttribute*		
		E	*IIR_SuccAttribute*		
		E	*IIR_PredAttribute*		
		E	*IIR_LeftOfAttribute*		
		E	*IIR_RightOfAttribute*		
		E	*IIR_RangeAttribute*		
		E	*IIR_ReverseRangeAttribute*		
		E	*IIR_LengthAttribute*		
		E	*IIR_DelayedAttribute*		
		E	*IIR_StableAttribute*		
		E	*IIR_QuietAttribute*		
		E	*IIR_TransactionAttribute*		
		E	*IIR_AscendingAttribute*		
		E	*IIR_EventAttribute*		
		E	*IIR_ActiveAttribute*		
		E	*IIR_LastEventAttribute*		
		E	*IIR_LastActiveAttribute*		
		E	*IIR_LastValueAttribute*		
		E	*IIR_DrivingAttribute*		
		E	*IIR_DrivingValueAttribute*		
		E	*IIR_SimpleNameAttribute*		
		E	*IIR_InstanceNameAttribute*		
		E	*IIR_PathNameAttribute*		
		E	*IIR_AcrossAttribute*		
		E	*IIR_ThroughAttribute*		
		E	*IIR_ReferenceAttribute*		
		E	*IIR_ContributionAttribute*		
		E	*IIR_Tolerance*		
		E	*IIR_DotAttribute*		
		E	*IIR_IntegAttribute*		
		E	*IIR_AboveAttribute*		

TABLE 168. Class hierarchy derived from IIR_Name class

		E	*IIR_ZOHAttribute*		
		E	*IIR_LTFAttribute*		
		E	*IIR_ZTFAttribute*		

12.1 IIR_Name

12.1.1 Derived Class Description

The pre-defined **IIR_Name** class represents the general class of refers to explicitly or implicitly declared named entities.

12.1.2 Properties

TABLE 169. IIR_Name Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS, Verilog
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR**
Predefined child classes	**IIR_SimpleName** **IIR_SelectedName** **IIR_SelectedNameByAll** **IIR_IndexedName** **IIR_SliceName** **IIR_Attribute** **IIR_EntityName**
Instantiation?	No, not directly allowed from this class
Application-specific data elements	None, not allowed for this class
Public data elements	None

12.1.3 Predefined Public Methods

All IIR_Name methods must be applied to a valid IIR_Name class and are atomic.

12.1.3.1 Static Name Lookup Methods

These methods translate an identifier or name into zero or more matching referents. If there is a single match, it returns as the first match. If there is more than one match, it returns as a dynamically allocated array of pointers to the matches; the caller must then deallocated this array.

```
static IIR_Declaration**
    lookup(IIR_TextLiteral*       identifier,
           IR_Int32&              number_of_matches,
           IIR_Declaration*&      first_match);
static IIR_Declaration**
    lookup(IIR_Name*              name,
           IR_Int32&              number_of_matches,
           IIR_Declaration*&      first_match);
```

12.1.3.2 Prefix Methods

These methods refer to the name’s prefix.

```
void
    set_prefix(    IIR*    prefix);
IIR*
    get_prefix();
```

12.2 IIR_SimpleName

12.2.1 Derived Class Description

The predefined **IIR_SimpleNam**e class represents an entity named by a simple IIR_TextLiteral.

12.2.2 Properties

TABLE 170. IIR_SelectedName Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS, Verilog
IR_Kind enumeration value	**IIR_SIMPLE_NAME**
Parent class	**IIR_Name**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.2.3 Predefined Public Methods

IIR_SimpleName methods other than the constructor must be applied to a valid IIR_SimpleName. All of the following methods are atomic.

12.2.3.1 Constructor Method

The constructor method initializes a selected name from an unspecified source location and an unspecified text literal.

```
    IIR_SimpleName();
```

12.2.3.2 Name Methods

These methods refer to the name's text literal.

```
void
    set_name(      IIR_TextLiteral*       name);
IIR_TextLiteral*
    get_name();
```

12.2.3.3 Destructor Method

The destructor method releases the name before deleting the object itself.

```
~IIR_SimpleName();
```

12.3 IIR_SelectedName

12.3.1 Derived Class Description

The predefined **IIR_SelectedNam**e class represents naming in which a prefix denotes a collection of entities and the suffix further specifies a subset of the collection.

12.3.2 Properties

TABLE 171. **IIR_SelectedName Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS, Verilog
IR_Kind enumeration value	**IR_SELECTED_NAME**
Parent class	**IIR_Name**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.3.3 Predefined Public Methods

IIR_SelectedName methods other than the constructor must be applied to a valid IIR_SelectedName. All of the following methods are atomic.

12.3.3.1 Constructor Method

The constructor method initializes a selected name from an unspecified source location, an unspecified prefix and an unspecified suffix.

```
IIR_SelectedName();
```

12.3.3.2 Parameters

These methods refer to the name's suffix.

```
void
    set_suffix(    IIR*            suffix);
IIR*
    get_suffix();
```

12.3.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_SelectedName();
```

12.4 IIR_SelectedNameByAll

12.4.1 Derived Class Description

The predefined **IIR_SelectedNameByAll** class represents all of the individual items present in the collection denoted by the prefix.

12.4.2 Properties

TABLE 172. IIR_SelectedNameByAll Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IIR_SELECTED_NAME_BY_ALL**
Parent class	**IIR_Name**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.4.3 Predefined Public Methods

IIR_SelectedNameByAll methods other than the constructor must be applied to a valid IIR_SelectedNameByAll. All of the following methods are atomic.

12.4.3.1 Constructor Method

The constructor method initializes a selected name from an unspecified source location and an unspecified prefix.

```
IIR_SelectedNameByAll();
```

12.4.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_SelectedNameByAll();
```

12.5 IIR_IndexedName

12.5.1 Derived Class Description

The predefined **IIR_IndexedName** denotes a single element of an array.

12.5.2 Properties

TABLE 173. **IIR_IndexedName Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS, Verilog
IR_Kind enumeration value	**IR_INDEXED_NAME**
Parent class	**IIR_Name**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.5.3 Predefined Public Methods

IIR_IndexedName methods other than the constructor must be applied to a valid IIR_IndexName. All of the following methods are atomic.

12.5.3.1 Constructor Method

The constructor method initializes an indexed name from an unspecified source location, unspecified prefix and unspecified suffix.

```
    IIR_IndexedName();
```

12.5.3.2 Parameters

These methods refer to the name's suffix (an expression which evaluates to a single integer).

```
void
    set_suffix(    IIR*          suffix);
IIR*
    get_suffix();
```

12.5.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_IndexedName();
```

12.6 IIR_SliceName

12.6.1 Derived Class Description

The predefined **IIR_SliceName** refers to zero or more elements of an array via a range.

12.6.2 Properties

TABLE 174. IIR_SliceName Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS, Verilog
IR_Kind enumeration value	**IR_SLICE_NAME**
Parent class	**IIR_Name**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.6.3 Predefined Public Methods

IIR_SliceName methods other than the constructor must be applied to a valid IIR_SliceName. All of the following methods are atomic.

12.6.3.1 Constructor Method

The constructor method initializes a selected name from an unspecified source location, an unspecified prefix and an unspecified suffix.

```
IIR_SliceName();
```

12.6.3.2 Parameters

These methods refer to the name's suffix (an expression which evaluates to a range).

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.6.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_SliceName();
```

12.7 IIR_Attribute

12.7.1 Derived Class Description

The predefined **IIR_Attribute** class refers to a value, function or implicitly named entity.

12.7.2 Properties

TABLE 175. **IIR_Attribute Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Name**

TABLE 175. **IIR_Attribute Properties**

Predefined child classes	**IIR_UserAttribute** **IIR_BaseAttribute** **IIR_LeftAttribute** **IIR_RightAttribute** **IIR_LowAttribute** **IIR_HighAttribute** **IIR_AscendingAttribute** **IIR_ImageAttribute** **IIR_ValueAttribute** **IIR_PosAttribute** **IIR_ValAttribute** **IIR_SuccAttribute** **IIR_PredAttribute** **IIR_LeftOfAttribute** **IIR_RightOfAttribute** **IIR_RangeAttribute** **IIR_ReverseRangeAttribute** **IIR_LengthAttribute** **IIR_DelayedAttribute** **IIR_StableAttribute** **IIR_QuietAttribute** **IIR_TransactionAttribute** **IIR_EventAttribute** **IIR_ActiveAttribute** **IIR_LastEventAttribute** **IIR_LastActiveAttribute** **IIR_LastValueAttribute** **IIR_DrivingAttribute** **IIR_DrivingValueAttribute** **IIR_SimpleNameAttribute** **IIR_InstanceNameAttribute** **IIR_PathNameAttribute** **IIR_AcrossAttribute** **IIR_ThroughAttribute** **IIR_ReferenceAttribute** **IIR_ContributionAttribute** **IIR_ToleranceAttribute** **IIR_DotAttribute** **IIR_IntegAttribute** **IIR_AboveAttribute** **IIR_ZOHAttribute** **IIR_LTFAttribute** **IIR_ZTFAttribute**
Instantiation?	No, not directly allowed from this class

TABLE 175. IIR_Attribute Properties

Application-specific data elements	None, not allowed for this class
Public data elements	None

12.8 IIR_UserAttribute

12.8.1 Derived Class Description

The predefined **IIR_UserAttribute** refers to a named value associated with an entity.

12.8.2 Properties

TABLE 176. IIR_UserAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_USER_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.8.3 Predefined Public Methods

IIR_UserAttribute methods other than the constructor must be applied to a valid IIR_UserAttribute. All of the following methods are atomic.

12.8.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_UserAttribute();
```

12.8.3.2 Parameters

These methods refer to the attribute's suffix (an attribute name).

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.8.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_UserAttribute();
```

12.9 IIR_BaseAttribute

12.9.1 Derived Class Description

The predefined **IIR_BaseAttribut**e refers to the base type of the attribute's prefix.

12.9.2 Properties

TABLE 177. IIR_BaseAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_BASE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.9.3 Predefined Public Methods

IIR_BaseAttribute methods other than the constructor must be applied to a valid IIR_BaseAttribute. All of the following methods are atomic.

12.9.3.1 Constructor Method

The constructor method initializes an attribute from a unspecified source location and unspecified prefix.

```
IIR_BaseAttribute();
```

12.9.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_BaseAttribute();
```

12.10 IIR_LeftAttribute

12.10.1 Derived Class Description

The predefined **IIR_LeftAttribute** class represents the left bound of a scalar object or the left bound of the Nth dimension of an array object.

12.10.2 Properties

TABLE 178. IIR_LeftAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_LEFT_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.10.3 Predefined Public Methods

IIR_LeftAttribute methods other than the constructor must be applied to a valid IIR_LeftAttribute. All of the following methods are atomic.

12.10.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
        IIR_LeftAttribute();
```

12.10.3.2 Parameters

These methods refer to the attribute's suffix.

```
    void
        set_suffix(    IIR*           suffix);
    IIR*
        get_suffix();
```

12.10.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_LeftAttribute();
```

12.11 IIR_RightAttribute

12.11.1 Derived Class Description

The predefined **IIR_RightAttribute** class represents the right bound of a scalar object or the right bound of the Nth dimension of an array object.

12.11.2 Properties

TABLE 179. IIR_RightAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_RIGHT_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.11.3 Predefined Public Methods

IIR_RightAttribute methods other than the constructor must be applied to a valid IIR_RightAttribute. All of the following methods are atomic.

12.11.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_RightAttribute();
```

12.11.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.11.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_RightAttribute();
```

12.12 IIR_LowAttribute

12.12.1 Derived Class Description

The **IIR_LowAttribute** predefined class represents the lower bound of an entity of scalar type (prefix).

12.12.2 Properties

TABLE 180. **IIR_LowAttribute Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_LOW_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.12.3 Predefined Public Methods

IIR_LowAttribute methods other than the constructor must be applied to a valid IIR_LowAttribute. All of the following methods are atomic.

12.12.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_LowAttribute();
```

12.12.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.12.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_LowAttribute();
```

12.13 IIR_HighAttribute

12.13.1 Derived Class Description

The **IIR_HighAttribute** predefined class represents the higher bound of an entity of scalar type (prefix).

12.13.2 Properties

TABLE 181. IIR_HighAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_HIGH_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.13.3 Predefined Public Methods

IIR_HighAttribute methods other than the constructor must be applied to a valid IIR_HighAttribute. All of the following methods are atomic.

12.13.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_HighAttribute();
```

12.13.3.2 Parameters

These methods refer to the attribute’s suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.13.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_HighAttribute();
```

12.14 IIR_AscendingAttribute

12.14.1 Derived Class Description

The predefined **IIR_AscendingAttribute** is true if the prefix is a scalar type or subtype having ascending range or the Nth dimension of an array object having the specified suffix.

12.14.2 Properties

TABLE 182. IIR_AscendingAttribute Properties

Applicable language(s)	VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_ASCENDING_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.14.3 Predefined Public Methods

IIR_AscendingAttribute methods other than the constructor must be applied to a valid IIR_AscendingAttribute. All of the following methods are atomic.

12.14.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_AscendingAttribute();
```

12.14.3.2 Parameters

These methods refer to the attribute’s suffix.

```
void
    set_suffix(    IIR*            suffix);
IIR*
    get_suffix();
```

12.14.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_AscendingAttribute();
```

12.15 IIR_ImageAttribute

12.15.1 Derived Class Description

The predefined **IIR_ImageAttribute** class represents the printable form of its suffix having type specified by the prefix.

12.15.2 Properties

TABLE 183. IIR_ImageAttribute Properties

Applicable language(s)	VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_IMAGE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.15.3 Predefined Public Methods

IIR_ImageAttribute methods other than the constructor must be applied to a valid IIR_ImageAttribute. All of the following methods are atomic.

12.15.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_ImageAttribute();
```

12.15.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR_ScalarTypeDefinition*     suffix);
IIR_ScalarTypeDefinition*
    get_suffix();
```

12.15.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_ImageAttribute();
```

12.16 IIR_ValueAttribute

12.16.1 Derived Class Description

The predefined **IIR_ValueAttribute** represents the value of the suffix interpreted via the type denoted by the prefix.

12.16.2 Properties

TABLE 184. IIR_ValueAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_VALUE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.16.3 Predefined Public Methods

IIR_ValueAttribute methods other than the constructor must be applied to a valid IIR_ValueAttribute. All of the following methods are atomic.

12.16.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
    IIR_ValueAttribute();
```

12.16.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.16.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_ValueAttribute();
```

12.17 IIR_PosAttribute

12.17.1 Derived Class Description

The predefined **IIR_PosAttribute** represents the position number of the suffix interpreted in terms of a type (or subtype) denoted by the prefix.

12.17.2 Properties

TABLE 185. IIR_PosAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_POS_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.17.3 Predefined Public Methods

IIR_PosAttribute methods other than the constructor must be applied to a valid IIR_PosAttribute. All of the following methods are atomic.

12.17.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_PosAttribute();
```

12.17.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR_ScalarTypeDefinition*     suffix);
IIR_ScalarTypeDefinition*
    get_suffix();
```

12.17.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_PosAttribute();
```

12.18 IIR_ValAttribute

12.18.1 Derived Class Description

The predefined **IIR_ValAttribute** represents the value, in terms of the type or subtype denoted by the prefix, of the suffix.

12.18.2 Properties

TABLE 186. IIR_ValAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_VAL_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.18.3 Predefined Public Methods

IIR_ValAttribute methods other than the constructor must be applied to a valid IIR_ValAttribute. All of the following methods are atomic.

12.18.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
        IIR_ValAttribute();
```

12.18.3.2 Parameters

These methods refer to the attribute's suffix.

```
    void
        set_suffix(    IIR_ScalarTypeDefinition*            suffix);
    IIR_ScalarTypeDefinition*
        get_suffix();
```

12.18.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name’s prefix or suffix before deleting the object itself.

```
~IIR_ValAttribute();
```

12.19 IIR_SuccAttribute

12.19.1 Derived Class Description

The predefined **IIR_SuccAttribute** class represents the value which is one greater than the suffix when interpreted using the type or subtype denoted by the prefix.

12.19.2 Properties

TABLE 187. IIR_SuccAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_SUCC_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.19.3 Predefined Public Methods

IIR_SuccAttribute methods other than the constructor must be applied to a valid IIR_SuccAttribute. All of the following methods are atomic.

12.19.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
    IIR_SuccAttribute();
```

12.19.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR_ScalarTypeDefinition*            suffix);
IIR_ScalarTypeDefinition*
    get_suffix();
```

12.19.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_SuccAttribute();
```

12.20 IIR_PredAttribute

12.20.1 Derived Class Description

The predefined **IIR_PredAttribute** class represents the value which is one less than the suffix when interpreted using the type or subtype denoted by the prefix.

12.20.2 Properties

TABLE 188. IIR_PredAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_PRED_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.20.3 Predefined Public Methods

IIR_PredAttribute methods other than the constructor must be applied to a valid IIR_PredAttribute. All of the following methods are atomic.

12.20.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_PredAttribute();
```

12.20.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR_ScalarTypeDefinition*             suffix);
IIR_ScalarTypeDefinition*
    get_suffix();
```

12.20.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name’s prefix or suffix before deleting the object itself.

```
~IIR_PredAttribute();
```

12.21 IIR_LeftOfAttribute

12.21.1 Derived Class Description

The predefined **IIR_LeftOfAttribute** class represents the value which is to the left of the suffix when interpreted using the type or subtype denoted by the prefix.

12.21.2 Properties

TABLE 189. IIR_LeftOfAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_LEFT_OF_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.21.3 Predefined Public Methods

IIR_LeftOfAttribute methods other than the constructor must be applied to a valid IIR_LeftOfAttribute. All of the following methods are atomic.

12.21.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
    IIR_LeftOfAttribute();
```

12.21.3.2 Parameters

These methods refer to the attribute’s suffix.

```
void
    set_suffix(    IIR_ScalarTypeDefinition*            suffix);
IIR_ScalarTypeDefinition*
    get_suffix();
```

12.21.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name’s prefix or suffix before deleting the object itself.

```
~IIR_LeftOfAttribute();
```

12.22 IIR_RightOfAttribute

12.22.1 Derived Class Description

The predefined **IIR_LeftOfAttribut**e class represents the value which is to the right of the suffix when interpreted using the type or subtype denoted by the prefix.

12.22.2 Properties

TABLE 190. IIR_RightOfAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_RIGHT_OF_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.22.3 Predefined Public Methods

IIR_RightOfAttribute methods other than the constructor must be applied to a valid IIR_RightOfAttribute. All of the following methods are atomic.

12.22.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_RightOfAttribute();
```

12.22.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR_ScalarTypeDefinition*            suffix);
IIR_ScalarTypeDefinition*
    get_suffix();
```

12.22.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_RightOfAttribute();
```

12.23 IIR_RangeAttribute

12.23.1 Derived Class Description

The predefined **IIR_RangeAttribute** represents the range of the Nth dimension (suffix) of the prefix (an array type, subtype or object).

12.23.2 Properties

TABLE 191. IIR_RangeAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IIR_RANGE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.23.3 Predefined Public Methods

IIR_RangeAttribute methods other than the constructor must be applied to a valid IIR_RangeAttribute. All of the following methods are atomic.

12.23.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
    IIR_RangeAttribute();
```

12.23.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*            suffix);
IIR*
    get_suffix();
```

12.23.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_RangeAttribute();
```

12.24 IIR_ReverseRangeAttribute

12.24.1 Derived Class Description

The predefined **IIR_ReverseRangeAttribute** represents the reverse range of the Nth dimension (suffix) of the prefix (an array type, subtype or object).

12.24.2 Properties

TABLE 192. IIR_ReverseRange Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_REVERSE_RANGE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.24.3 Predefined Public Methods

IIR_ReverseRangeAttribute methods other than the constructor must be applied to a valid IIR_ReverseRangeAttribute. All of the following methods are atomic.

12.24.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_ReverseRangeAttribute();
```

12.24.3.2 Parameters

These methods refer to the attribute’s suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.24.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_ReverseRangeAttribute();
```

12.25 IIR_LengthAttribute

12.25.1 Derived Class Description

The predefined **IIR_LengthAttribute** represents the number of elements present in the Nth dimension (suffix) of an array object, type or subtype denoted by the prefix.

12.25.2 Properties

TABLE 193. IIR_Length Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_LENGTH_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.25.3 Predefined Public Methods

IIR_LengthAttribute methods other than the constructor must be applied to a valid IIR_LengthAttribute. All of the following methods are atomic.

12.25.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_LengthAttribute();
```

12.25.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.25.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_LengthAttribute();
```

12.26 IIR_DelayedAttribute

12.26.1 Derived Class Description

The predefined **IIR_DelayedAttribute** class represents the delayed form of a signal (prefix) where the suffix denotes the delay value

12.26.2 Properties

TABLE 194. IIR_DelayedAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_DELAYED_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.26.3 Predefined Public Methods

IIR_DelayedAttribute methods other than the constructor must be applied to a valid IIR_DelayedAttribute. All of the following methods are atomic.

12.26.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
        IIR_DelayedAttribute();
```

12.26.3.2 Parameters

These methods refer to the attribute’s suffix.

```
    void
        set_suffix(    IIR*                    suffix);
    IIR*
        get_suffix();
```

12.26.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_DelayedAttribute();
```

12.27 IIR_StableAttribute

12.27.1 Derived Class Description

The predefined **IIR_StableAttribute** class represents a boolean asserting that an event has not occurred on a signal (denoted by the prefix) for an interval of time denoted by the optional suffix.

12.27.2 Properties

TABLE 195. IIR_StableAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_STABLE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.27.3 Predefined Public Methods

IIR_StableAttribute methods other than the constructor must be applied to a valid IIR_StableAttribute. All of the following methods are atomic.

12.27.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_StableAttribute();
```

12.27.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.27.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_StableAttribute();
```

12.28 IIR_QuietAttribute

12.28.1 Derived Class Description

The predefined **IIR_QuietAttribute** represents a boolean denoting that a signal (the prefix) has been quiet for at least the time interval denoted by the suffix.

12.28.2 Properties

TABLE 196. IIR_QuietAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_QUIET_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.28.3 Predefined Public Methods

IIR_QuietAttribute methods other than the constructor must be applied to a valid IIR_QuietAttribute. All of the following methods are atomic.

12.28.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
    IIR_QuietAttribute();
```

12.28.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.28.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_QuietAttribute();
```

12.29 IIR_TransactionAttribute

12.29.1 Derived Class Description

The predefined **IIR_TransactionAttribute** represents a boolean which toggles value on each simulation cycle where a signal (denoted by the prefix) becomes active.

12.29.2 Properties

TABLE 197. **IIR_TransactionAttribute Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_TRANSACTION_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.29.3 Predefined Public Methods

IIR_TransactionAttribute methods other than the constructor must be applied to a valid IIR_TransactionAttribute. All of the following methods are atomic.

12.29.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_TransactionAttribute();
```

12.29.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_TransactionAttribute();
```

12.30 IIR_EventAttribute

12.30.1 Derived Class Description

The predefined **IIR_EventAttribute** class is a boolean attribute representing when an event has occurred during the current simulation cycle on a signal denoted by the attribute's prefix.

12.30.2 Properties

TABLE 198. IIR_EventAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IIR_EVENT_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.30.3 Predefined Public Methods

IIR_EventAttribute methods other than the constructor must be applied to a valid IIR_EventAttribute. All of the following methods are atomic.

12.30.3.1 Constructor Method

The constructor method initializes an attribute from a unspecified source location, and unspecified prefix.

```
IIR_EventAttribute();
```

12.30.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_EventAttribute();
```

12.31 IIR_ActiveAttribute

12.31.1 Derived Class Description

The predefined **IIR_ActiveAttribute** class represents a boolean attribute denoting if a signal (denoted by the prefix) is active on the current simulation cycle.

12.31.2 Properties

TABLE 199. IIR_ActiveAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_ACTIVE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.31.3 Predefined Public Methods

IIR_ActiveAttribute methods other than the constructor must be applied to a valid IIR_ActiveAttribute. All of the following methods are atomic.

12.31.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_ActiveAttribute();
```

12.31.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_ActiveAttribute();
```

12.32 IIR_LastEventAttribute

12.32.1 Derived Class Description

The predefined **IIR_LastEventAttribute** class represents the interval of time since the last event occurred on a signal denoted by the prefix.

12.32.2 Properties

TABLE 200. IIR_LastEventAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_LAST_EVENT_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.32.3 Predefined Public Methods

IIR_LastEventAttribute methods other than the constructor must be applied to a valid IIR_LastEventAttribute. All of the following methods are atomic.

12.32.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_LastEventAttribute();
```

12.32.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_LastEventAttribute();
```

12.33 IIR_LastActiveAttribute

12.33.1 Derived Class Description

The predefined **IIR_LastActiveAttribut**e class represents the amount of time since a signal denoted by the prefix was last active.

12.33.2 Properties

TABLE 201. IIR_LastActiveAttribute Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_LAST_ACTIVE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.33.3 Predefined Public Methods

IIR_LastActiveAttribute methods other than the constructor must be applied to a valid IIR_LastActiveAttribute. All of the following methods are atomic.

12.33.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_LastActiveAttribute();
```

12.33.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_LastActiveAttribute();
```

12.34 IIR_LastValueAttribute

12.34.1 Derived Class Description

The predefined **IIR_LastValueAttribute** class represents the last value assumed by a signal denoted by the attribute's prefix.

12.34.2 Properties

TABLE 202. **IIR_LastValueAttribute Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_LAST_VALUE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.34.3 Predefined Public Methods

IIR_LastValueAttribute methods other than the constructor must be applied to a valid IIR_LastValueAttribute. All of the following methods are atomic.

12.34.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, unspecified prefix and unspecified suffix.

```
IIR_LastValueAttribute();
```

12.34.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_LastValueAttribute();
```

12.35 IIR_DrivingAttribute

12.35.1 Derived Class Description

The predefined **IIR_DrivingAttribute** class assists in determining which driver is driving a signal denoted by the attribute prefix.

12.35.2 Properties

TABLE 203. IIR_DrivingAttribute Properties

Applicable language(s)	VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_DRIVING_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.35.3 Predefined Public Methods

IIR_DrivingAttribute methods other than the constructor must be applied to a valid IIR_DrivingAttribute. All of the following methods are atomic.

12.35.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_DrivingAttribute();
```

12.35.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_DrivingAttribute();
```

12.36 IIR_DrivingValueAttribute

12.36.1 Derived Class Description

The predefined **IIR_DrivingAttribute** class assists in determining which driver is driving a signal denoted by the attribute prefix.

12.36.2 Properties

TABLE 204. IIR_DrivingValueAttribute Properties

Applicable language(s)	VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IIR_DRIVING_VALUE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.36.3 Predefined Public Methods

IIR_DrivingAttribute methods other than the constructor must be applied to a valid IIR_DrivingAttribute. All of the following methods are atomic.

12.36.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_DrivingValueAttribute();
```

12.36.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_DrivingValueAttribute();
```

12.37 IIR_SimpleNameAttribute

12.37.1 Derived Class Description

The predefined **IIR_InstanceNameAttribute** class represents the simple name associated with a named entity.

12.37.2 Properties

TABLE 205. IIR_SimpleNameAttribute Properties

Applicable language(s)	VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_SIMPLE_NAME_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.37.3 Predefined Public Methods

IIR_SimpleNameAttribute methods other than the constructor must be applied to a valid IIR_SimpleNameAttribute. All of the following methods are atomic.

12.37.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_SimpleNameAttribute();
```

12.37.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix or suffix before deleting the object itself.

```
~IIR_SimpleNameAttribute();
```

12.38 IIR_PathNameAttribute

12.38.1 Derived Class Description

The predefined **IIR_PathNameAttribute** class represents the hierarchical path name associated with a named entity excluding its simple name.

12.38.2 Properties

TABLE 206. IIR_PathNameAttribute Properties

Applicable language(s)	VHDL-93, VHDL-98*, VHDL-AMS
IR_Kind enumeration value	**IR_PATH_NAME_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.38.3 Predefined Public Methods

IIR_PathNameAttribute methods other than the constructor must be applied to a valid IIR_PathNameAttribute. All of the following methods are atomic.

12.38.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_PathNameAttribute( );
```

12.38.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_PathNameAttribute();
```

12.39 IIR_AcrossAttribute

12.39.1 Derived Class Description

The predefined **IIR_AcrossAttribute** class represents the across type of the nature prefix.

12.39.2 Properties

TABLE 207. IIR_AcrossAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_ACROSS_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.39.3 Predefined Public Methods

IIR_AcrossAttribute methods other than the constructor must be applied to a valid IIR_AcrossAttribute. All of the following methods are atomic.

12.39.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_AcrossAttribute(   );
```

12.39.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_AcrossAttribute();
```

12.40 IIR_ThroughAttribute

12.40.1 Derived Class Description

The predefined **IIR_ThroughAttribute** class represents the through type of the nature prefix.

12.40.2 Properties

TABLE 208. **IIR_ThroughAttribute Properties**

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_THROUGH_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.40.3 Predefined Public Methods

IIR_ThroughAttribute methods other than the constructor must be applied to a valid IIR_ThroughAttribute. All of the following methods are atomic.

12.40.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_ThroughAttribute(  );
```

12.40.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name’s prefix before deleting the object itself.

```
~IIR_ThroughAttribute();
```

12.41 IIR_ReferenceAttribute

12.41.1 Derived Class Description

The predefined **IIR_ReferenceAttribute** class represents the reference terminal for the nature denoted by the prefix.

12.41.2 Properties

TABLE 209. IIR_ReferenceAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_REFERENCE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.41.3 Predefined Public Methods

IIR_ReferenceAttribute methods other than the constructor must be applied to a valid IIR_ReferenceAttribute. All of the following methods are atomic.

12.41.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_ReferenceAttribute();
```

12.41.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_ReferenceAttribute();
```

12.42 IIR_ContributionAttribute

12.42.1 Derived Class Description

The predefined **IIR_ContributionAttribute** class represents the through quantity formed by the prefix (plus terminal) and a minus terminal which is the reference of the nature of the prefix.

12.42.2 Properties

TABLE 210. IIR_ContributionAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_CONTRIBUTION_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.42.3 Predefined Public Methods

IIR_ContributionAttribute methods other than the constructor must be applied to a valid IIR_ContributionAttribute. All of the following methods are atomic.

12.42.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_ContributionAttribute();
```

12.42.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_ContributionAttribute();
```

12.43 IIR_ToleranceAttribute

12.43.1 Derived Class Description

The predefined **IIR_ToleranceAttribute** class represents the tolerance of the the prefix.

12.43.2 Properties

TABLE 211. IIR_ToleranceAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_TOLERANCE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.43.3 Predefined Public Methods

IIR_ToleranceAttribute methods other than the constructor must be applied to a valid IIR_ToleranceAttribute. All of the following methods are atomic.

12.43.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_ToleranceAttribute();
```

12.43.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name’s prefix before deleting the object itself.

```
~IIR_ToleranceAttribute();
```

12.44 IIR_DotAttribute

12.44.1 Derived Class Description

The predefined **IIR_DotAttribute** class represents the time differential of the prefix when the attribute is evaluated.

12.44.2 Properties

TABLE 212. IIR_DotAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_DOT_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.44.3 Predefined Public Methods

IIR_DotAttribute methods other than the constructor must be applied to a valid IIR_DotAttribute. All of the following methods are atomic.

12.44.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_DotAttribute(      );
```

12.44.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_DotAttribute();
```

12.45 IIR_IntegAttribute

12.45.1 Derived Class Description

The predefined **IIR_IntegAttribute** class represents the time integral of the prefix from time = 0 to the time at which evaluation takes place.

12.45.2 Properties

TABLE 213. IIR_IntegAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_INTEG_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.45.3 Predefined Public Methods

IIR_IntegAttribute methods other than the constructor must be applied to a valid IIR_IntegAttribute. All of the following methods are atomic.

12.45.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_IntegAttribute(    );
```

12.45.3.2 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_IntegAttribute();
```

12.46 IIR_AboveAttribute

12.46.1 Derived Class Description

The predefined **IIR_AboveAttribute** class represents a boolean tolerance function which is TRUE if the prefix is sufficiently above the suffix, FALSE if it is sufficiently below and undefined in between.

12.46.2 Properties

TABLE 214. IIR_AboveAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_ABOVE_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.46.3 Predefined Public Methods

IIR_AboveAttribute methods other than the constructor must be applied to a valid IIR_AboveAttribute. All of the following methods are atomic.

12.46.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
    IIR_AboveAttribute(    );
```

12.46.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.46.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_AboveAttribute();
```

12.47 IIR_ZOHAttribute

12.47.1 Derived Class Description

The predefined **IIR_ZTFAttribute** class represents a sampled signal.

12.47.2 Properties

TABLE 215. IIR_ZOHAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_ZOH_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.47.3 Predefined Public Methods

IIR_ZOHAttribute methods other than the constructor must be applied to a valid IIR_ZOHAttribute. All of the following methods are atomic.

12.47.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
    IIR_ZOHAttribute(      );
```

12.47.3.2 Parameters

These methods refer to the attribute's suffix.

```
void
    set_suffix(    IIR*           suffix);
IIR*
    get_suffix();
```

12.47.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_ZOHAttribute();
```

12.48 IIR_LTFAttribute

12.48.1 Derived Class Description

The predefined **IIR_LTFAttribute** class represents the Laplace transfer function.

12.48.2 Properties

TABLE 216. **IIR_LTFAttribute Properties**

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_LTF_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.48.3 Predefined Public Methods

IIR_LTFAttribute methods other than the constructor must be applied to a valid IIR_LTFAttribute. All of the following methods are atomic.

12.48.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
    IIR_LTFAttribute(      );
```

12.48.3.2 Parameter Methods

These methods refer to the attribute's numerator coefficients and denominator coefficients.

```
void
    set_num(       IIR*          num);
IIR*
    get_num();
void
    set_den(       IIR*          den);
IIR*
```

```
get_den();
```

12.48.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
~IIR_LTFAttribute();
```

12.49 IIR_ZTFAttribute

12.49.1 Derived Class Description

The predefined **IIR_ZTFAttribute** class represents the Z-domain transfer function.

12.49.2 Properties

TABLE 217. IIR_ZIFAttribute Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_ZTF_ATTRIBUTE**
Parent class	**IIR_Attribute**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

12.49.3 Predefined Public Methods

IIR_ZTFAttribute methods other than the constructor must be applied to a valid IIR_ZTFAttribute. All of the following methods are atomic.

12.49.3.1 Constructor Method

The constructor method initializes an attribute from an unspecified source location, and unspecified prefix.

```
IIR_ZTFAttribute(     );
```

12.49.3.2 Parameters

```
These methods refer to the attribute's suffix.
void
    set_num(       IIR*           num);
IIR*
    get_num();
void
    set_den(       IIR*           den);
IIR*
    get_den();
void
    set_t(         IIR*           t);
```

```
IIR*
    get_t();
void
    set_initial_delay(IIR*        initial_delay);
IIR*
    get_initial_delay();
```

12.49.3.3 Destructor Method

The destructor method deletes any other names and releases any canonical objects present in the name's prefix before deleting the object itself.

```
    ~IIR_ZTFAttribute();
```

CHAPTER 13 *IIR_Expression Derived Classes*

This chapter specifies the properties, predefined public methods and predefined public data associated with IIR_Expression and all predefined classes derived from IIR_Expression (as shown in Table 218 on page 415). All derivative classes of IIR_Expression are dynamically and individually allocated.

Monadic and dyadic operators, both language-defined and overloaded, may be described by either a specific operator class or more generally by a function call.

TABLE 218. Class derivation hierarchy from IIRBase_Expression

	Level 3 Derived Classes (only classes derived from IIRBase_Expression)	E	Level 4 Derived Classes	E	Level 5 Derived Classes
E	*IIR_MonadicOperator*	E	*IIR_IdentityOperator*		
		E	*IIR_NegationOperator*		
		E	*IIR_AbsoluteOperator*		
		E	*IIR_NotOperator*		
E	*IIR_DyadicOperator*	E	*IIR_AndOperator*		
		E	*IIR_OrOperator*		
		E	*IIR_NandOperator*		
		E	*IIR_NorOperator*		
		E	*IIR_XorOperator*		
		E	*IIR_XnorOperator*		
		E	*IIR_EqualityOperator*		
		E	*IIR_InequalityOperator*		
		E	*IIR_LessThanOperator*		
		E	*IIR_LessThanOrEqualOperator*		

TABLE 218. Class derivation hierarchy from IIRBase_Expression

		E	*IIR_GreaterThanOperator*		
		E	*IIR_GreaterThanOrEqualOperator*		
		E	*IIR_SLLOperator*		
		E	*IIR_SRLOperator*		
		E	*IIR_SLAOperator*		
		E	*IIR_SRAOperator*		
		E	*IIR_ROLOperator*		
		E	*IIR_ROROperator*		
		E	*IIR_AdditionOperator*		
		E	*IIR_SubtractionOperator*		
		E	*IIR_ConcatentationOperator*		
		E	*IIR_MultiplicationOperator*		
		E	*IIR_DivisionOperator*		
		E	*IIR_ModulusOperator*		
		E	*IIR_RemainderOperator*		
		E	*IIR_ExponentiationOperator*		
E	*IIR_FunctionCall*				
E	*IIR_PhysicalLiteral*				
E	*IIR_Aggregate*				
E	*IIR_OthersInitialization*				
E	*IIR_QualifiedExpression*				
E	*IIR_TypeConversion*				
E	*IIR_Allocator*				

13.1 IIR_Expression

13.1.1 Derived Class Description

The predefined **IIR_Expression** class and its derivatives represent formulas for computing a value. They may appear in a wide variety of contexts including type definitions, the initial value of declarations, and as parameters within sequential or concurrent statements.

13.1.2 Properties

TABLE 219. IIR_Expression Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR**
Predefined child classes	IIR_MonadicOperator IIR_DyadicOperator IIR_FunctionCall IIR_PhysicalLiteral IIR_Aggregate IIR_OthersInitialization IIR_QualifiedExpression IIR_TypeConversion IIR_Allocator
Instantiation?	Indirectly via any of the derived classes of IIR_Expression
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_Expression class
Public data elements	None

13.1.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_Expression. All of the following methods are atomic.

13.1.3.1 Subtype Methods

All expressions have a well defined type. In some cases the subtype will be an unconstrained base type, it others it is a constrained subtype.

```
void
    set_subtype(            IIR_TypeDefinition*            subtype);
IIR_TypeDefinition*
    get_subtype();
```

13.2 IIIR_MonadicOperator

13.2.1 Derived Class Description

The predefined **IIR_MonadicOperator** operators include identity, negation, absolute value and not. Derivatives of this class represent both language predefined monadic operators and subprograms defining overloadings of these operators.

13.2.2 Properties

TABLE 220. **IIR_MonadicOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None: not directly instantiated
Parent class	**IIR_Expression**
Predefined child classes	**IIR_IdentityOperator** **IIR_NegationOperator** **IIR_AbsoluteOperator** **IIR_NotOperator**
Instantiation?	Indirectly via any of the derived classes of IIR_MonadicOperator
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_MonadicOperator class
Public data elements	None

13.2.3 Predefined Public Methods

13.2.3.1 Subprogram Implementation Methods

The subprogram methods denote a subprogram declaration representing implementation of this operator instance.

```
void
    set_implementation(    IIR_SubprogramDeclaration*            implementation);
IIR_SubprogramDeclaration*
    get_implementation();
```

13.2.3.2 Operand

Monadic operators utilize a single operand.

```
void
    set_operand(            IIR*             operand);
IIR*
    get_operand();
```

13.3 IIR_IdentityOperator

13.3.1 Derived Class Description

The predefined **IIR_IdentityOperator** class represents the identity operator and its overloadings.

13.3.2 Properties

TABLE 221. **IIR_IdentityOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_IDENTITY_OPERATOR**
Parent class	**IIR_MonadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.3.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_IdentityOperator object. All of the following methods are atomic.

13.3.3.1 Constructor Method

The constructor initializes an IIR_IdentityOperator object.

```
IIR_IdentityOperator();
```

13.3.3.2 Destructor Method

The destructor deletes the operand, and subtype before deleting the operator object itself.

```
~IIR_IdentityOperator();
```

13.4 IIR_NegationOperator

13.4.1 Derived Class Description

The predefined **IIR_NegationOperator** class represents the negation operator and its overloadings.

13.4.2 Properties

TABLE 222. **IIR_NegationOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_NEGATION_OPERATOR**
Parent class	**IIR_MonadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.4.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_NegationOperator object. All of the following methods are atomic.

13.4.3.1 Constructor Method

The constructor initializes an IIR_NegationOperator object.

```
IIR_NegationOperator();
```

13.4.3.2 Destructor Method

The destructor deletes the operand, and subtype before deleting the operator object itself.

```
~IIR_NegationOperator();
```

13.5 IIR_AbsoluteOperator

13.5.1 Derived Class Description

The predefined **IIR_AbsoluteOperator** class represents the absolute operator and its overloadings.

13.5.2 Properties

TABLE 223. **IIR_AbsoluteOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ABSOLUTE_OPERATOR**
Parent class	IIR_MonadicOperator
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.5.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AbsoluteOperator object. All of the following methods are atomic.

13.5.3.1 Constructor Method

The constructor initializes an IIR_AbsoluteOperator object.

```
IIR_AbsoluteOoperator();
```

13.5.3.2 Destructor Method

The destructor deletes the operand, and subtype before deleting the operator object itself.

```
~IIR_AbsoluteOperator();
```

13.6 IIR_NotOperator

13.6.1 Derived Class Description

The predefined **IIR_NotOperator** class represents the logical NOT operator and its overloadings.

13.6.2 Properties

TABLE 224. IIR_NotOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_NOT_OPERATOR**
Parent class	**IIR_MonadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.6.3 Predefined Public Method

Except for the constructor, all of the following methods must be applied to a valid IIR_NotOperator object. All of the following methods are atomic.

13.6.3.1 Constructor Method

The constructor initializes an IIR_NotOperator object.

```
IIR_NotOperator()
```

13.6.4 Destructor Method

The destructor deletes the operand, and subtype before deleting the operator object itself.

```
~IIR_NotOperator();
```

13.7 IIR_DyadicOperator

13.7.1 Derived Class Description

The predefined **IIR_DyadicOperator** classes include logical, relational, shift, adding, multiplying and miscellaneous operators. Derivatives of this class represent both language predefined dyadic operators and subprograms defining overloadings of these operators.

13.7.2 Properties

TABLE 225. IIR_DyadicOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	None: not directly instantiated
Parent class	**IIR_Expression**
Predefined child classes	**IIR_AndOperator** **IIR_OrOperator** **IIR_NandOperator** **IIR_NorOperator** **IIR_XorOperator** **IIR_XnorOperator** **IIR_EqualityOperator** **IIR_InequalityOperator** **IIR_LessThanOperator** **IIR_LessThanOrEqualOperator** **IIR_GreaterThanOperator** **IIR_GreaterThanOrEqualOperator** **IIR_SLLOperator** **IIR_SLAOperator** **IIR_SRLOperator** **IIR_SRAOperator** **IIR_ROLOperator** **IIR_ROROperator** **IIR_AdditionOperator** **IIR_SubtractionOperator** **IIR_ConcatentationOperator** **IIR_MultiplicationOperator** **IIR_DivisionOperator** **IIR_ModulusOperator** **IIR_RemainderOperator** **IIR_ExponentiationOperator**
Instantiation?	Indirectly via any of the derived classes of IIR_DyadicOperator
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_DyadicOperator class
Public data elements	None

13.7.3 Predefined Public Methods

13.7.3.1 Subprogram Implementation Methods

The subprogram methods denote a subprogram declaration representing implementation of this operator instance.

```
void
    set_implementation(    IIR_SubprogramDeclaration*            implementation);
IIR_SubprogramDeclaration*
    get_implementation();
```

13.7.3.2 Operand Methods

The dyadic operators have left and right operands.

```
void
    set_left_operand(      IIR*            left_operand);
IIR*
    get_left_operand();
void
    set_right_operand(     IIR*            right_operand);
IIR*
    get_right_operand();
```

13.8 IIR_AndOperator

13.8.1 Derived Class Description

The predefined **IIR_AndOperator** class represents the logical AND operator and its overloadings.

13.8.2 Properties

TABLE 226. IIR_AndOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98, VHDL-AMS
IR_Kind enumeration value	**IR_AND_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.8.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AndOperator object. All of the following methods are atomic.

13.8.3.1 Constructor Method

The constructor initializes an IIR_AndOperator object.

```
IIR_AndOperator();
```

13.8.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_AndOperator();
```

13.9 IIR_OrOperator

13.9.1 Derived Class Description

The predefined IIR_OrOperator class represents the logical OR operator and its overloadings.

13.9.2 Properties

TABLE 227. **IIR_OrOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_OR_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.9.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_OrOperator object. All of the following methods are atomic.

13.9.3.1 Constructor Method

The constructor initializes an IIR_OrOperator object.

```
IIR_OrOperator();
```

13.9.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_OrOperator();
```

13.10 IIR_NandOperator

13.10.1 Derived Class Description

The predefined **IIR_NandOperator** class represents the logical NAND operator and its overloadings.

13.10.2 Properties

TABLE 228. IIR_NandOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_NAND_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.10.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_NandOperator object. All of the following methods are atomic.

13.10.3.1 Constructor Method

The constructor initializes an IIR_NandOperator object.

```
IIR_NandOperator();
```

13.10.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_NandOperator();
```

13.11 IIR_NorOperator

13.11.1 Derived Class Description

The predefined IIR_NorOperator class represents the logical NOR operator and its overloadings.

13.11.2 Properties

TABLE 229. IIR_NorOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_NOR_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.11.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_NorOperator object. All of the following methods are atomic.

13.11.3.1 Constructor Method

The constructor initializes an IIR_NorOperator object.

```
IIR_NorOperator();
```

13.11.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_NorOperator();
```

13.12 IIR_XorOperator

13.12.1 Derived Class Description

The predefined **IIR_XorOperator** class represents the logical XOR operator and its overloadings.

13.12.2 Properties

TABLE 230. **IIR_XorOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_XOR_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.12.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_XorOperator object. All of the following methods are atomic.

13.12.3.1 Constructor Method

The constructor initializes an IIR_XorOperator object.

```
IIR_XorOperator();
```

13.12.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_XorOperator();
```

13.13 IIR_XnorOperator

13.13.1 Derived Class Description

The predefined **IIR_XnorOperator** class represents the logical XNOR operator and its overloadings.

13.13.2 Properties

TABLE 231. IIR_XnorOperator Properties

Applicable language(s)	VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_XNOR_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.13.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_XnorOperator object. All of the following methods are atomic.

13.13.3.1 Constructor Method

The constructor initializes an IIR_XnorOperator object.

```
IIR_XnorOperator();
```

13.13.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_XnorOperator();
```

13.14 IIR_EqualityOperator

13.14.1 Derived Class Description

The predefined **IIR_EqualityOperator** class represents the relational equality operator and its overloadings.

13.14.2 Properties

TABLE 232. **IIR_EqualityOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_EQUALITY_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.14.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_EqualityOperator object. All of the following methods are atomic.

13.14.3.1 Constructor Method

The constructor initializes an IIR_EqualityOperator object.

```
IIR_EqualityOperator();
```

13.14.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_EqualityOperator();
```

13.15 IIR_InequalityOperator

13.15.1 Derived Class Description

The predefined **IIR_InequalityOperato**r class represents the relational inequality operator and its overloadings.

13.15.2 Properties

TABLE 233. **IIR_InequalityOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_INEQUALITY_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.15.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_InequalityOperator object. All of the following methods are atomic.

13.15.3.1 Constructor Method

The constructor initializes an IIR_InequalityOperator object.

```
IIR_InequalityOperator();
```

13.15.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_InequalityOperator();
```

13.16 IIR_LessThanOperator

13.16.1 Derived Class Description

The predefined **IIR_LessThanOperator** class represents the relational less than operator and its overloadings.

13.16.2 Properties

TABLE 234. IIR_LessThanOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_LESS_THAN_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.16.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_LessThanOperator object. All of the following methods are atomic.

13.16.3.1 Constructor Method

The constructor initializes an IIR_LessThanOperator object.

```
IIR_LessThanOperator();
```

13.16.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_LessThanOperator();
```

13.17 IIR_LessThanOrEqualOperator

13.17.1 Derived Class Description

The predefined **IIR_LessThanOrEqualOperator** class represents the relational less than or equal operator and its overloadings.

13.17.2 Properties

TABLE 235. IIR_LessThanOrEqualOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_LESS_THAN_OR_EQUAL_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.17.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_LessThanOrEqualOperator object. All of the following methods are atomic.

13.17.3.1 Constructor Method

The constructor initializes an IIR_LessThanOrEqualOperator object.

```
IIR_LessThanOrEqualOperator();
```

13.17.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_LessThanOrEqualOperator();
```

13.18 IIR_GreaterThanOperator

13.18.1 Derived Class Description

The **IIR_GreaterThanOperator** class represents the relational greater than operator and its overloadings.

13.18.2 Properties

TABLE 236. IIR_GreaterThanOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_GREATER_THAN_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.18.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_GreaterThanOperator object. All of the following methods are atomic.

13.18.3.1 Constructor Method

The constructor initializes an IIR_GreaterThanOperator object.

```
IIR_GreaterThanOperator();
```

13.18.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_GreaterThanOperator();
```

13.19 IIR_GreaterThanOrEqualOperator

13.19.1 Derived Class Description

The predefined **IIR_GreaterThanOrEqualOperator** class represents the relational less than or equal operator and its overloadings.

13.19.2 Properties

TABLE 237. IIR_GreaterThanOrEqualOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_GREATER_THAN_OR_EQUAL_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.19.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_GreaterThanOrEqualOperator object. All of the following methods are atomic.

13.19.3.1 Constructor Method

The constructor initializes an IIR_GreaterThanOrEqualOperator object.

```
IIR_GreaterThanOrEqualOoperator();
```

13.19.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_GreaterThanOrEqualOperator();
```

13.20 IIR_SLLOperator

13.20.1 Derived Class Description

The predefined **IIR_SLLOperator** class represents the shift left logical operator and its overloadings.

13.20.2 Properties

TABLE 238. **IIR_SLLOperator Properties**

Applicable language(s)	VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SLL_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.20.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SLLOperator object. All of the following methods are atomic.

13.20.3.1 Constructor Method

The constructor initializes an IIR_SLLOperator object.

```
IIR_SLLOoperator();
```

13.20.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_SLLOperator();
```

13.21 IIR_SRLOperator

13.21.1 Derived Class Description

The predefined **IIR_SRLOperator** class represents the shift right logical operator and its overloadings.

13.21.2 Properties

TABLE 239. IIR_SRLOperator Properties

Applicable language(s)	VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SRL_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.21.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SRLOperator object. All of the following methods are atomic.

13.21.3.1 Constructor Method

The constructor initializes an IIR_SRLOperator object.

```
IIR_SRLOoperator();
```

13.21.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_SRLOperator();
```

13.22 IIR_SLAOperator

13.22.1 Derived Class Description

The predefined **IIR_SLAOperator** class represents the shift left arithmetic operator and its overloadings.

13.22.2 Properties

TABLE 240. IIR_SLAOperator Properties

Applicable language(s)	VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_SLA_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.22.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SLAOperator object. All of the following methods are atomic.

13.22.3.1 Constructor Method

The constructor initializes an IIR_SLAOperator object.

```
IIR_SLAOoperator();
```

13.22.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_SLAOperator();
```

13.23 IIR_SRAOperator

13.23.1 Derived Class Description

The predefined **IIR_SRAOperator** class represents the shift right arithmetic operator and its overloadings.

13.23.2 Properties

TABLE 241. IIR_SRAOperator Properties

Applicable language(s)	VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SRA_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.23.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SRAOperator object. All of the following methods are atomic.

13.23.3.1 Constructor Method

The constructor initializes an IIR_SRAOperator object.

```
IIR_SRAOoperator();
```

13.23.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_SRAOperator();
```

13.24 IIR_ROLOperator

13.24.1 Derived Class Description

The predefined IIR_ROLOperator class represents the rotate right logical operator and its overloadings.

13.24.2 Properties

TABLE 242. IIR_ROLOperator Properties

Applicable language(s)	VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ROL_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.24.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ROLOperator object. All of the following methods are atomic.

13.24.3.1 Constructor Method

The constructor initializes an IIR_ROLOperator object.

```
IIR_ROLOoperator();
```

13.24.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_ROLOperator();
```

13.25 IIR_ROROperator

13.25.1 Derived Class Description

The predefined **IIR_ROROperator** class represents the rotate right logical operator and its overloadings.

13.25.2 Properties

TABLE 243. IIR_ROROperator Properties

Applicable language(s)	VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ROR_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.25.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ROROperator object. All of the following methods are atomic.

13.25.3.1 Constructor Method

The constructor initializes an IIR_ROROperator object.

```
IIR_ROROoperator();
```

13.25.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_ROROperator();
```

13.26 IIR_AdditionOperator

13.26.1 Derived Class Description

The predefined IIR_AdditionOperator class represents the addition operator and its overloadings.

13.26.2 Properties

TABLE 244. IIR_AdditionOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ADDITION_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.26.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AdditionOperator object. All of the following methods are atomic.

13.26.3.1 Constructor Method

The constructor initializes an IIR_AdditionOperator object.

```
IIR_AdditionOperator();
```

13.26.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_AdditionOperator();
```

13.27 IIR_SubtractionOperator

13.27.1 Derived Class Description

The predefined **IIR_SubtractionOperator** class represents the subtraction operator and its overloadings.

13.27.2 Properties

TABLE 245. IIR_SubtractionOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SUBTRACTION_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.27.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SubtractionOperator object. All of the following methods are atomic.

13.27.3.1 Constructor Method

The constructor initializes an IIR_SubtractionOperator object.

```
IIR_SubtractionOperator();
```

13.27.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_SubtractionOperator();
```

13.28 IIR_ConcatenationOperator

13.28.1 Derived Class Description

The predefined **IIR_ConcatenationOperator** class represents the concatenation operator and its overloadings.

13.28.2 Properties

TABLE 246. **IIR_ConcatenationOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONCATENATION_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.28.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConcatenationOperator object. All of the following methods are atomic.

13.28.3.1 Constructor Method

The constructor initializes an IIR_ConcatenationOperator object.

```
IIR_ConcatenationOperator();
```

13.28.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_ConcatenationOperator();
```

13.29 IIR_MultiplicationOperator

13.29.1 Derived Class Description

The predefined **IIR_MultiplicationOperator** class represents the multiplication operator and its overloadings.

13.29.2 Properties

TABLE 247. IIR_MultiplicationOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_MULTIPLICATION_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.29.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_MultiplicationOperator object. All of the following methods are atomic.

13.29.3.1 Constructor Method

The constructor initializes an IIR_MultiplicationOperator object.

```
IIR_MultiplicationOperator();
```

13.29.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_MultiplicationOperator();
```

13.30 IIR_DivisionOperator

13.30.1 Derived Class Description

The predefined **IIR_DivisionOperator** class represents the division operator and its overloadings.

13.30.2 Properties

TABLE 248. IIR_DivisionOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_DIVISION_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.30.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_DivisionOperator object. All of the following methods are atomic.

13.30.3.1 Constructor Method

The constructor initializes an IIR_DivisionOperator object.

```
IIR_DivisionOperator();
```

13.30.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_DivisionOperator();
```

13.31 IIR_ModulusOperator

13.31.1 Derived Class Description

The predefined **IIR_ModulusOperator** class represents the modulus operator and its overloadings.

13.31.2 Properties

TABLE 249. IIR_ModulusOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_MODULUS_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.31.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ModulusOperator object. All of the following methods are atomic.

13.31.3.1 Constructor Method

The constructor initializes an IIR_ModulusOperator object.

```
IIR_ModulusOperator();
```

13.31.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_ModulusOperator();
```

13.32 IIR_RemainderOperator

13.32.1 Derived Class Description

The predefined **IIR_RemainderOperator** class represents the remainder operator and its overloadings.

13.32.2 Properties

TABLE 250. **IIR_RemainderOperator Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_REMAINDER_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.32.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_RemainderOperator object. All of the following methods are atomic.

13.32.3.1 Constructor Method

The constructor initializes an IIR_RemainderOperator object.

```
IIR_RemainderOperator();
```

13.32.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_RemainderOperator();
```

13.33 IIR_ExponentiationOperator

13.33.1 Derived Class Description

The predefined **IIR_ExponentiationOperator** class represents the exponentiation operator and its overloadings.

13.33.2 Properties

TABLE 251. IIR_ExponentiationOperator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_EXPONENTIATION_OPERATOR**
Parent class	**IIR_DyadicOperator**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.33.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ExponentiationOperator object. All of the following methods are atomic.

13.33.3.1 Constructor Method

The constructor initializes an IIR_ExponentiationOperator object.

```
IIR_ExponentiationOperator();
```

13.33.3.2 Destructor Method

The destructor deletes the left operand, right operand and subtype before deleting the operator object itself.

```
~IIR_ExponentiationOperator();
```

13.34 IIR_PhysicalLiteral

13.34.1 Derived Class Description

The predefined **IIR_PhysicalLiteral** class represents a value formed by multiplying an abstract literal and unit name, resulting in a value of physical type.

13.34.2 Properties

TABLE 252. IIR_PhysicalLiteral Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_PHYSICAL_LITERAL**
Parent class	**IIR_Expression**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.34.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_PhysicalLiteral object. All of the following methods are atomic.

13.34.3.1 Constructor Method

The constructor initializes a physical literal object:

```
    IIR_PhysicalLiteral();
```

13.34.3.2 Abstract Literal Methods

The physical unit's abstract literal defines the unit name's multiplier.

```
void
    set_abstract_literal(       IIR*           abstract_literal);
IIR*
    get_abstract_literal();
```

13.34.3.3 Unit Name Methods

The unit name implies some multiple of a physical type's primary unit.

```
void
    set_unit_name(        IIR_PhysicalUnit*             unit);
IIR_PhysicalUnit*
    get_unit_name();
```

13.34.3.4 Destructor Method

Destruction of the physical literal involves destruction of the abstract literal and unit name. Note that the abstract literal is typically a canonical object, and thus is not actually deallocated. In a like fashion, the actual secondary unit is not actually deallocated.

```
    ~IIR_PhysicalLiteral();
```

13.35 IIR_Aggregate

13.35.1 Derived Class Description

The predefined **IIR_Aggregate** class combines one or more values into a composite value having a record or array type.

13.35.2 Properties

TABLE 253. IIR_Aggregate Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_AGGREGATE**
Parent class	**IIR_Expression**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

13.35.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_Aggregate object. All of the following methods are atomic.

13.35.3.1 Constructor Method

The constructor initializes an IIR_Aggregate object:

```
IIR_Aggregate();
```

13.35.3.2 Destructor Method

The destructor method deletes each of the aggregate elements before deleting the aggregate object itself.

```
~IIR_Aggregate();
```

13.35.4 Predefined Public Data

IIR_Aggregates include a single public data element:

```
IIR_AssociationList       element_association_list
```

13.36 IIR_OthersInitialization

13.36.1 Derived Class Description

The predefined **IIR_OthersInitialization** class defines the value associated with elements not explicitly specified.

13.36.2 Properties

TABLE 254. IIR_OthersInitialization Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_OTHERS_INITIALIZATION**
Parent class	**IIR_Expression**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.36.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_OthersInitialization object. All of the following methods are atomic.

13.36.3.1 Constructor Method

The constructor initializes an IIR_OthersInitialization object:

```
IIR_OthersInitialization();
```

13.36.3.2 Expression Methods

The expression methods refer to the value associated with elements which were not otherwise specified.

```
void
    set_expression(        IIR*           v);
IIR*
    get_expression();
```

13.36.3.3 Destructor Method

The destructor method first deletes the expression, then the others initialization object itself.

```
~IIR_OthersInitialization();
```

13.37 IIR_FunctionCall

13.37.1 Derived Class Description

The predefined **IIR_FunctionCall** classes elaborate and execute a subprogram call, resulting in a returned value.

13.37.2 Properties

TABLE 255. IIR_FunctionCall Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_FUNCTION_CALL**
Parent class	**IIR_Expression**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

13.37.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_FunctionCall object. All of the following methods are atomic.

13.37.3.1 Constructor Method

The constructor initializes an IIR_FunctionCall object:

```
    IIR_FunctionCall();
```

13.37.3.2 Subprogram Implementation Methods

The subprogram methods denote a subprogram declaration representing the function being called.

```
void
    set_implementation(    IIR_SubprogramDeclaration*            implementation);
IIR_SubprogramDeclaration*
    get_implementation();
```

13.37.3.3 Destructor Method

```
void
    ~IIR_FunctionCall();
```

13.37.4 Predefined Public Data

The IIR_FunctionCall class includes the following predefined public data:

```
IIR_AssociationList        parameter_association_list
```

13.38 IIR_QualifiedExpression

13.38.1 Derived Class Description

The predefined **IIR_QualifiedExpression** class make the type or subtype of an expression explicit.

13.38.2 Properties

TABLE 256. IIR_QualifiedExpression Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_QUALIFIED_EXPRESSION**
Parent class	**IIR_Expression**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.38.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_QualifiedExpression object. All of the following methods are atomic.

13.38.3.1 Constructor Method

The constructor initializes a qualified expression object:

```
    IIR_QualifiedExpression();
```

13.38.3.2 Type Mark Methods

Type mark methods denote the explicit type or subtype of the qualified expression.

```
void
    set_type_mark(        IIR_TypeDefinition*          type_mark);
IIR_TypeDefinition*
    get_type_mark();
```

13.38.3.3 Expression Methods

The expression methods refer to the expression or aggregate who’s type or subtype is being specified.

```
void
    set_expression(IIR*           expression);
IIR*
    get_expression();
```

13.38.3.4 Destructor Method

The destructor method first deletes the expression, then the qualified expression object itself.

```
void
    ~IIR_QualifiedExpression();
```

13.39 IIR_TypeConversion

13.39.1 Derived Class Description

The predefined **IIR_TypeConversion** classes provide for explicit conversion between closely related types.

13.39.2 Properties

TABLE 257. IIR_TypeConversion Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_TYPE_CONVERSION**
Parent class	**IIR_Expression**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.39.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_TypeConversion object. All of the following methods are atomic.

13.39.3.1 Constructor Method

The constructor initializes a qualified expression object:

```
IIR_TypeConversion();
```

13.39.3.2 Type Mark Methods

Type mark methods denote the explicit type or subtype of the type conversion expression.

```
void
    set_type_mark(        IIR_TypeDefinition*                  type_mark);
IIR_TypeDefinition*
    get_type_mark();
```

13.39.3.3 Expression Methods

The expression methods refer to the expression who’s type or subtype is being specified.

```
void
    set_expression(        IIR*            expression);
IIR*
    get_expression();
```

13.39.3.4 Destructor Method

The destructor method first deletes the expression, then the type conversion object itself.

```
    ~IIR_TypeConversion();
```

13.40 IIR_Allocator

13.40.1 Derived Class Description

The predefined **IIR_Allocator** class dynamically allocates an object of specified subtype.

13.40.2 Properties

TABLE 258. IIR_Allocator Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_ALLOCATOR**
Parent class	**IIR_Expression**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

13.40.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_Allocator object. All of the following methods are atomic.

13.40.3.1 Constructor Method

The constructor initializes an allocator object:

```
IIR_Allocator();
```

13.40.3.2 Type Mark Methods

Type mark methods denote the subtype of the object to be allocated.

13.40.3.3 void *set_type_mark*(IIR_TypeDefinition*type_mark); IIR_TypeDefinition* *get_type_mark*();

13.40.3.4 Value Methods

A value may be associated with an allocator expression denoting the initial value associated with objects upon allocation.

```
void
    set_value(IIR*          value);
IIR*
   get_value();
```

13.40.3.5 Destructor Method

The destructor method first deletes the expression, then the allocator object itself.

```
    ~IIR_Allocator();
```

CHAPTER 14

IIR_SequentialStatement Derived Classes

This chapter specifies the properties, predefined public methods and predefined public data associated with IIR_SequentialStatement and all predefined classes derived from IIR_SequentialStatement (as shown in Table 259 on page 467). All derivative classes of IIR_SequentialStatement are dynamically and individually allocated.

Sequential statements generally appear (directly or indirectly) within a process, procedure or function. Some language-mandated restrictions further constrain the location of sequential statements. Sequential statements are generally referenced from a sequential statement list, the next method of a sequential statement or as the target of a control-flow sequential statement.

Application-specific data elements may be added to an optional extension class layer just after differentiating the different kinds of sequential statements. For example, these extensions may provide additional debugging information or information specific to a particular backend. Application-specific methods may be added to extension layers anywhere in the IIR class hierarchy. New application-specific classes may be created from either the IIR_SequentialStatement or one of its derivatives.

TABLE 259. Class hierarchy derived from IIR_SequentialStatement class

	Level 4 Derived Classes (only classes derived from IIR_SequentialStatement)	**E**	**Level 5 Derived Classes**	**E**	**Level 6 Derived Classes**
E	*IIR_WaitStatement*				
E	*IIR_AssertionStatement*				
E	*IIR_ReportStatement*				
E	*IIR_SignalAssignmentStatement*				
E	*IIR_VariableAssignmentStatement*				
E	*IIR_ProcedureCallStatement*				
E	*IIR_IfStatement*				

TABLE 259. Class hierarchy derived from IIR_SequentialStatement class

E	*IIR_CaseStatement*				
E	*IIR_ForLoopStatement*				
E	*IIR_WhileLoopStatement*				
E	*IIR_NextStatement*				
E	*IIR_ExitStatement*				
E	*IIR_ReturnStatement*				
E	*IIR_NullStatement*				
E	*IIR_BreakStatement*				

14.1 IIR_SequentialStatement

14.1.1 Derived Class Description

The predefined **IIR_SequentialStatement** classes specify individual sequential statements within a process or subprogram.

14.1.2 Properties

TABLE 260. **IIR_SequentialStatement Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Statement**
Predefined child classes	**IIR_WaitStatement** **IIR_AssertionStatement** **IIR_ReportStatement** **IIR_SignalAssignmentStatement** **IIR_VariableAssignmentStatement** **IIR_ProcedureCallStatement** **IIR_IfStatement** **IIR_CaseStatement** **IIR_ForLoopStatement** **IIR_WhileLoopStatement** **IIR_NextStatement** **IIR_ExitStatement** **IIR_ReturnStatement** **IIR_NullStatement** **IIR_BreakStatement**
Instantiation?	Indirectly via any of the derived classes of IIR_SequentialStatement
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_SequentialStatement class
Predefined public data elements	None

14.2 IIR_WaitStatement

14.2.1 Derived Class Description

The **IIR_WaitStatement** suspends execution pending a signal event, boolean condition and/or time out interval. Such statements may appear almost anywhere a sequential statement may appear (some restrictions in subprograms).

14.2.2 Properties

TABLE 261. IIR_WaitStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_WAIT_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

14.2.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SequentialStatement object. All of the following methods are atomic.

14.2.3.1 Constructor Method

The constructor initializes a sequential statement object:

```
IIR_WaitStatement();
```

14.2.3.2 Condition Clause Methods

The condition clause must evaluate to TRUE in order for execution to proceed beyond a wait statement. If no condition clause is associated with the wait statement, the pointer to condition clause returns NIL. Consistently, assigning a NIL value to the condition clause disassociates any condition clause from the wait statement.

```
void
    set_condition_clause(        IIR*            condition_clause);
IIR*
    get_condition_clause();
```

14.2.3.3 Timeout Clause Methods

The maximum length of time a wait statement may suspend execution is given by the timeout clause. A NIL value for the clause denotes timeout at STD.STANDARD.TIME’HIGH.

```
void
    set_timeout_clause(          IIR*          timeout_clause);
IIR*
    get_timeout_clause();
```

14.2.3.4 Destructor Method

The destructor method deletes the label (if any), sensitivity list (if any), condition clause (if any), timeout clause (if any), then deletes the wait statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
    ~IIR_WaitStatement();
```

14.2.4 Predefined Public Data

The IIR_WaitStatement’s predefined public data includes:

```
IIR_DesignatorList               sensitivity_list;
```

14.3 IIR_AssertionStatement

14.3.1 Derived Class Description

The predefined **IIR_AssertionStatement** checks that a specified condition is true. If the condition is false, a report is made with specified severity level. This statement may appear anywhere a sequential statement may appear.

14.3.2 Properties

TABLE 262. **IIR_AssertionStatement Properties**

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_ASSERTION_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

14.3.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_AssertionStatement object. All of the following methods are atomic.

14.3.3.1 Constructor Method

The constructor initializes an assertion statement object.

```
    IIR_AssertionStatement();
```

14.3.3.2 Assertion Argument Methods

Assertion argument methods reference the assertion condition, report expression and severity expression used by the sequential assertion statement.

```
void
    set_assertion_condition(      IIR*           assertion_condition);
IIR*
    get_assertion_condition();
```

```
void
    set_report_expression(        IIR*          report_expression):
IIR*
    get_report_expression();
void
    set_severity_expression(      IIR*          expression);
IIR*
    get_severity_expression();
```

14.3.3.3 Destructor Method

The destructor method deletes the label (if any), condition expression, report expression (if any), and severity expression (if any) before deleting the assertion statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
    ~IIR_AssertionStatement();
```

14.4 IIR_ReportStatement

14.4.1 Derived Class Description

The predefined **IIR_ReportStatement** responds with a message at a specified severity level. Such statements may appear anywhere a sequential statement may appear.

14.4.2 Properties

TABLE 263. IIR_ReportStatement Properties

Applicable language(s)	VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_REPORT_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

14.4.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ReportStatement object. All of the following methods are atomic.

14.4.3.1 Constructor Method

The constructor initializes a reports statement object.

```
IIR_ReportStatement();
```

14.4.3.2 Report Argument Methods

Assertion argument methods reference the assertion condition, report expression and severity expression used by the sequential assertion statement.

```
void
    set_report_expression(        IIR*            report_expression):
IIR*
    get_report_expression();
void
    set_severity_expression(      IIR*            severity_expression);
```

```
IIR*
    get_severity_expression();
```

14.4.3.3 Destructor Method

The destructor method deletes the label (if any), report expression (if any), and severity expression (if any) before deleting the report statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
void
    ~IIR_ReportStatement();
```

14.5 IIR_SignalAssignmentStatement

14.5.1 Derived Class Description

The predefined **IIR_SignalAssignmentStatement** updates the projected waveform output of one or more signal drivers. Such statements may appear anywhere a sequential statement may appear.

14.5.2 Properties

TABLE 264. IIR_SequentialStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IIR_SIGNAL_ASSIGNMENT_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

14.5.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SequentialStatement object. All of the following methods are atomic.

14.5.3.1 Constructor Method

The constructor initializes a signal assignment statement object.

```
    IIR_SignalAssignmentStatement();
```

14.5.3.2 Target Methods

Target methods refer to the target of a signal assignment statement.

```
void
    set_target(    IIR*           target);
IIR*
    get_target();
```

14.5.3.3 Delay Mechanism Methods

A signal assignment statement either uses transport or inertial delay.

```
void
    set_delay_mechanism(   IIR_DelayMechanism     delay_mechanism);
IIR_DelayMechanism
    get_delay_mechanism();
```

14.5.3.4 Reject Time Methods

If an inertial paradigm is selected, a signal assignment statement may optionally designate a rejection time expression, which applies to the entire waveform. If a transport paradigm is used to a signal assignment statement, the reject time expression must be NIL.

```
void
    set_reject_time_expression(   IIR*            reject_time_expression);
IIR*
    get_reject_time_expression();
```

14.5.3.5 Destructor Method

The destructor method deletes the label (if any), target, reject time expression (if any) and waveform before deleting the signal assignment statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
void
    ~IIR_SignalAssignmentStatement();
```

14.5.4 Predefined Public Data

The IIR_SignalAssignmentStatement's predefined public data includes the following data elements directly instantiated within the concurrent procedure call statement:

```
    IIR_WaveformList                   waveform;
```

14.6 IIR_VariableAssignmentStatement

14.6.1 Derived Class Description

The **IIR_VariableAssignmentStatement** updates the value of a variable with the value specified in an expression. Such statements may appear anywhere a sequential statement may appear.

14.6.2 Properties

TABLE 265. IIR_VariableAssignmentStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_VARIABLE_ASSIGNMENT_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

14.6.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_VariableAssignmentStatement object. All of the following methods are atomic.

14.6.3.1 Constructor Method

The constructor initializes a concurrent assertion statement object.

```
    IIR_VariableAssignmentStatement();
```

14.6.3.2 Target Methods

Target methods refer to the target of the variable assignment statement.

```
void
    set_target(    IIR*            target);
IIR*
    get_target();
```

14.6.3.3 Expression Methods

Expression methods refer to the value which is to be assigned to the target. Both target and expression must have the same type.

```
void
    set_expression(        IIR*           target);
IIR*
    get_expression();
```

14.6.3.4 Destructor Method

The destructor method deletes the label (if any), target, and right-hand-side expression before deleting the variable assignment statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
    ~IIR_VariableAssignmentStatement();
```

14.7 IIR_ProcedureCallStatement

14.7.1 Derived Class Description

The predefined **IIR_ProcedureCallStatement** dynamically elaborates and executes a procedure declaration. It may appear anywhere sequential statements are allowed.

14.7.2 Properties

TABLE 266. IIR_ProcedureCallStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IIR_PROCEDURE_CALL_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

14.7.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ProcedureCallStatement object. All of the following methods are atomic.

14.7.3.1 Constructor Method

The constructor initializes a concurrent conditional signal assignment object.

```
IIR_ProcedureCallStatement();
```

14.7.3.2 Procedure Methods

Procedure name methods reference an existing procedure declaration.

```
void
    set_procedure_name(    IIR*           procedure_name);
IIR*
    get_procedure_name();
```

14.7.3.3 Destructor Method

The destructor method deletes the label (if any), procedure name, and association list before deleting the procedure call statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
~IIR_ProcedureCallStatement();
```

14.7.4 Predefined Public Data

The IIR_ProcedureCallStatement's predefined public data includes the following data elements directly instantiated within the procedure call statement:

```
IIR_AssociationList              actual_parameter_part;
```

14.8 IIR_IfStatement

14.8.1 Derived Class Description

The predefined **IIR_IfStatement** provides for the optional, selective execution of one or more sequential statement lists. Such statements may appear anywhere sequential statements are allowed.

The IIR_IfStatement uses a chain of IIR_Elsif tuples to contain the elseif parts of the If statement. The IIR_Elsif tuple combines a condition and a sequence of statements to execute if the condition is true. If the recursion does not encounter a TRUE. The final else sequence of statements is the else_sequence in IIR_IfStatement.

14.8.2 Properties

TABLE 267. IIR_IfStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_IF_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

14.8.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_IfStatement object. All of the following methods are atomic.

14.8.3.1 Constructor Method

The constructor initializes an if statement object.

```
IIR_IfStatement();
```

14.8.3.2 Condition Methods

Condition methods refer to an expression of boolean type which is evaluated in order to determine which sequential statements are to be executed.

```
void
    set_condition(        IIR*    condition);
IIR*
    get_condition();
```

14.8.3.3 Elsif Methods

```
void
    set_elsif(    IIR_Elsif*    condition);
IIR_Elsif*
    get_elsif();
```

14.8.3.4 Destructor Method

The destructor method deletes the label (if any), condition, then statement sequence and else statement sequence before deleting the if statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
    ~IIR_IfStatement();
```

14.8.4 Predefined Public Data

The IIR_IfStatement's predefined public data includes the following data elements directly instantiated within the if statement:

```
IIR_SequentialStatementList      then_sequence;
IIR_SequentialStatementList      else_sequence;
```

14.9 IIR_CaseStatement

14.9.1 Derived Class Description

The predefined IIR_CaseStatement provides for execution of at most one sequential statement list from a set of alternatives. Such statements may appear anywhere sequential statements are allowed.

14.9.2 Properties

TABLE 268. IIR_CaseStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_CASE_STATEMEN**T
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

14.9.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_CaseStatement object. All of the following methods are atomic.

14.9.3.1 Constructor Method

The constructor initializes a component instantiation statement object.

```
    IIR_CaseStatement();
```

14.9.3.2 Expression Methods

The case statement expression is evaluated in order to select exactly one choice and implied sequence of statements to execute.

```
void
    set_expression(        IIR*              expression);
IIR*
    get_expression();
```

14.9.3.3 Destructor Method

The destructor method deletes the label (if any), (dispatch) expression, and case statement alternative list before deleting the case statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
~IIR_CaseStatement();
```

14.9.4 Predefined Public Data

The IIR_CaseStatement's predefined public data includes the following data elements directly instantiated within the case statement:

```
IIR_CaseStatementAlternativeList         case_statement_alternatives;
```

14.10 IIR_ForLoopStatement

14.10.1 Derived Class Description

The predefined **IIR_ForLoopStatement** executes a sequences of statements zero or more times, advancing the value of an iterator constant once before each execution of the loop body. Such statements may appear anywhere a sequential statement is allowed.

14.10.2 Properties

TABLE 269. IIR_ConcurrentGenerateForStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_FOOR_LOOP_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

14.10.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ForLoopStatement object. All of the following methods are atomic.

14.10.3.1 Constructor Method

The constructor initializes a concurrent generate for statement object.

```
IIR_ForLoopStatement();
```

14.10.3.2 Iteration Scheme Methods

The iteration scheme, a constant declaration, is the for loop iterator. The declaration’s subtype determines the iteration direction and range.

```
void
    set_iteration_scheme(  IIR_ConstantDeclaration*      iterator);
IIR_ConstantDeclaration*
    get_iteration_scheme();
```

14.10.3.3 Destructor Method

The destructor method deletes the label (if any), for iteration scheme, and loop body statements before deleting the loop statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
~IIR_ForLoopStatement();
```

14.10.4 Predefined Public Data

The IIR_ForLoopStatement's predefined public data includes the following data elements directly instantiated within the statement:

```
IIR_SequentialStatementList     sequence_of_statements;
IIR_DeclarationList             loop_declarations;
```

14.11 IIR_WhileLoopStatement

14.11.1 Derived Class Description

The predefined **IIR_WhileLoopStatement** executes a sequential statement list zero or more times. A boolean condition evaluates once before each iteration. If the condition evaluates true, the enclosed statement sequence executes, otherwise execute continues with the statement following the while loop statement.

14.11.2 Properties

TABLE 270. IIR_WhileLoopStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_WHILE_LOOP_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

14.11.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_WhileLoopStatement object. All of the following methods are atomic.

14.11.3.1 Constructor Method

The constructor initializes a while loop statement object.

```
    IIR_WhileLoopStatement();
```

14.11.3.2 While Condition Methods

The while condition is evaluated at the beginning of each iteration through the loop statement's body. When the while condition evaluates False, the loop execution terminates.

```
void
    set_while_codition(    IIR*            while_condition);
IIR*
    get_while_condition();
```

14.11.3.3 Destructor Method

The destructor method deletes the label (if any), while condition, and loop body statements before deleting the loop statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
~IIR_WhileLoopStatement();
```

14.11.4 Predefined Public Data

The IIR_WhileLoopStatement's predefined public data includes the following data elements directly instantiated within the statement:

```
IIR_SequentialStatementList     sequence_of_statements;
IIR_DeclarationList             loop_declarations;
```

14.12 IIR_NextStatement

14.12.1 Derived Class Description

The predefined **IIR_NextStatement** conditionally terminates execution of an enclosing loop iteration, potentially advancing to another iteration of the loop. Next statements may appear anywhere within an enclosing for or while loop.

14.12.2 Properties

TABLE 271. IIR_NextStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_NEXT_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

14.12.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_NextStatement object. All of the following methods are atomic.

14.12.3.1 Constructor Method

The constructor initializes a next statement object.

```
    IIR_NextStatement();
```

14.12.3.2 Enclosing Loop Methods

The enclosing loop methods designate the loop statement to which the next statement applies. Note that the loop statement to which the next statement refers may or may not have a label.

```
void
    set_enclosing_loop(    IIR_SequentialStatement*              loop);
IIR_SequentialStatement*
    get_enclosing_loop();
```

14.12.3.3 Condition Methods

The condition must evaluate to True in order for the next statement to transfer the flow of control. If the condition is NIL, control always transfers via the next statement.

```
void
    set_condition(        IIR*          condition);
IIR*
    get_condition();
```

14.12.3.4 Destructor Method

The destructor method deletes the label (if any), and condition before deleting the next statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
    ~IIR_NextStatement();
```

14.13 IIR_ExitStatement

14.13.1 Derived Class Description

The predefined **IIR_ExitStatement** conditionally terminates (all) iterations of an enclosing loop statement. Such statements may appear anywhere within an enclosing for or while loop.

14.13.2 Properties

TABLE 272. IIR_ExitStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_EXIT_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

14.13.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ExitStatement object. All of the following methods are atomic.

14.13.3.1 Constructor Method

The constructor initializes a concurrent generate if statement object.

```
IIR_ExitStatement();
```

14.13.3.2 Enclosing Loop Methods

The enclosing loop methods designate the loop statement to which the next statement applies. Note that the loop statement to which the next statement refers may or may not have a label.

```
void
    set_enclosing_loop(    IIR_SequentialStatement*      enclosing_loop);
IIR_SequentialStatement*
    get_enclosing_loop();
```

14.13.3.3 Condition Methods

The condition must evaluate to True in order for the exit statement to transfer the flow of control. If the condition is NIL, control always transfers via the exit statement.

```
void
    set_condition( IIR*           condition);
IIR*
    get_condition();
```

14.13.3.4 Destructor Method

The destructor method deletes the label (if any), and condition before deleting the exit statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
    ~IIR_ExitStatement();
```

14.14 IIR_ReturnStatement

14.14.1 Derived Class Description

The predefined **IIR_ReturnStatement** terminates execution of the inner-most enclosing subprogram body. Such statements may appear anywhere within a subprogram body.

14.14.2 Properties

TABLE 273. IIR_Return Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_RETURN_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

14.14.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ReturnStatement object. All of the following methods are atomic.

14.14.3.1 Constructor Method

The constructor initializes a return statement object.

```
    IIR_ReturnStatement();
```

14.14.3.2 Enclosing Subprogram Methods

The enclosing subprogram methods designate the subprogram declaration to which the return statement applies.

```
void
    set_enclosing_subprogram(    IIR_SubprogramDeclaration*    enclosing_subprogram);
IIR_SubprogramDeclaration*
    get_enclosing_subprogram();
```

14.14.3.3 Return Expression Methods

The return expression denotes the value to be returned with the return executes. Procedures always have a NIL return expression. Functions must have a non-NIL return expression matching the base type of the function's return type.

```
void
    set_return_expression(        IIR*           return_expression);
IIR*
    get_return_expression();
```

14.14.3.4 Destructor Method

The destructor method deletes the label (if any), and return expression before deleting the return statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
    ~IIR_ReturnStatement();
```

14.15 IIR_NullStatement

14.15.1 Derived Class Description

The predefined **IIR_NullSatement**s statements have no behavior. They act as a place-holder where one or more sequential statements are syntactically required, however no behavioral action is required. Such statements may appear anywhere a sequential statement is allowed.

14.15.2 Properties

TABLE 274. IIR_NullStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-A
IR_Kind enumeration value	**IR_NULL_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

14.15.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_NullStatement object. All of the following methods are atomic.

14.15.3.1 Constructor Method

The constructor initializes a null statement object.

```
IIR_NullStatement();
```

14.15.3.2 Destructor Method

The destructor method deletes the label (if any), and return expression before deleting the return statement object itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting storage associated with the canonical object.

```
void
    ~IIR_NullStatement();
```

14.16 IIR_BreakStatement

14.16.1 Derived Class Description

The predefined **IIR_BreakSatement** class indicates a break in the continuity of one or more quantities.

14.16.2 Properties

TABLE 275. IIR_BreakStatement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_BREAK_STATEMENT**
Parent class	**IIR_SequentialStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

14.16.3 Predefined Public Methods

14.16.3.1 Constructor Method

```
    IIR_BreakStatement();
```

14.16.3.2 Condition Method

```
void
    set_condition(IIR*     condition);
IIR*
    get_condition();
```

14.16.3.3 Destructor Method

```
void
    ~IIR_BreakStatement();
```

14.16.4 Predefined Public Data Elements

```
IIR_BreakList              break_list;
```

CHAPTER 15

IIR_ConcurrentStatement Derived Classes

This chapter specifies the properties, predefined public methods and predefined public data associated with the IIR_ConcurrentStatement class and all predefined classes derived from IIR_ConcurrentStatement (as shown in Table 276 on page 499). All derivative classes of IIR_ConcurrentStatement are dynamically and individually allocated.

Application-specific data elements may be added to an optional extension class layer just after differentiating the different kinds of concurrent statements. For example, these extensions may provide additional debugging information or information specific to a particular backend. Application-specific methods may be added to extension layers anywhere in the IIR class hierarchy. New application-specific classes may be created from either the IIR_ConcurrentStatement or one of its derivatives.

TABLE 276. Class hierarchy derived from IIR_ConcurrentStatement

	Level 4 Derived Classes (only classes derived from IIR_ConcurrentStatement)	**E**	**Level 5 Derived Classes**	**E**	**Level 6 Derived Classes**
E	*IIR_BlockStatement*				
E	*IIR_ProcessStatement*	E	*IIR_SensitizedProcessStatement*		
E	*IIR_ConcurrentProcedureCallStatement*				
E	*IIR_ConcurrentAssertionStatement*				
E	*IIR_ConcurrentConditionalSignalAssignmentt*				
E	*IIR_ConcurrentSelectedSignaAssignmentt*				
E	*IIR_ComponentInstantiationStatement*				
E	*IIR_ConcurrentGenerateForStatement*				
E	*IIR_ConcurrentGenerateIfStatement*				

15.1 IIR_ConcurrentStatement

15.1.1 Derived Class Description

The concurrent statement derived classes are all derived via the **IIR_ConcurrentStatement** class. These statements introduce one or more (concurrent) declarative regions containing one or more sequential execution threads.

15.1.2 Properties

TABLE 277. IIR_ConcurrentStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98, VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Statement**
Predefined child classes	**IIR_BlockStatement** **IIR_ProcessStatement** **IIR_ConcurrentProcedureCallStatement** **IIR_ConcurrentAssertionStatement** **IIR_ConcurrentConditionalSignalAssignment** **IIR_ConcurrentSelectedSignalAssignment** **IIR_ComponentInstantiationStatement** **IIR_ConcurrentGenerateForStatement** **IIR_ConcurrentGenerateIfStatement**
Instantiation?	Indirectly via any of the derived classes of IIR_ConcurrentStatement
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_ConcurrentStatement class
Public data elements	None

15.1.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid object having a class derived from IIR_ConcurrentStatement. All of the following methods are atomic.

15.2 IIR_BlockStatement

15.2.1 Derived Class Description

The predefined **IIR_BlockStatemen**t class introduces a single, concurrent, declarative region into the structure of a VHDL design unit. Block statements are dynamically allocated as a concurrent statement within architectures, other block statements and generate statements.

15.2.2 Properties

TABLE 278. IIR_BlockStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98, VHDL-AMS
IR_Kind enumeration value	**IR_BLOCK_STATEMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.2.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_BlockStatement object. All of the following methods are atomic.

15.2.3.1 Constructor Methods

The constructors initialize a block statement object from an unspecified source location, an unspecified declarator, no generic clause items, no generic map items, no port clause items, no port map items, no block declarative items and no block statements.

```
    IIR_BlockStatement();
```

15.2.3.2 Guard Expression Methods

Guard expressions determine the value of the block's optional, implicit guard signal. If a guard expression is transformed into the equivalent guard signal during analysis or elaboration, the guard expression becomes NIL. In a like manner, deleting the implicit guard signal from the block's declarative region also sets the guard expression to NIL.

```
void
    set_guard_expression( IIR*           guard_expression);
```

```
IIR*
    get_guard_expression();
```

15.2.3.3 Destructor Method

The list destructor method deletes the guard expression, generic clause, generic_map_aspect, port_clause, port_map_aspect, block_declarative_part and block_statement_part, then deletes the block statement object itself.

```
void
    ~IIR_BlockStatement();
```

15.2.4 Predefined Public Data

The IIR_BlockStatement's predefined public data elements include:

```
IIR_GenericList                  generic_clause;
IIR_AssociationList              generic_map_aspect;
IIR_PortList                     port_clause;
IIR_AssociationList              port_map_aspect;
IIR_DeclarationList              block_declarative_part;
IIR_ConcurrentStatementList      block_statement_part;
```

15.3 IIR_ProcessStatement

15.3.1 Derived Class Description

The predefiend **IIR_ProcessStatement** class represents a sequential declarative region and single thread of execution Such processes must appear within an architecture, concurrent block statement or concurrent generate statement.

15.3.2 Properties

TABLE 279. IIR_ProcessStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_PROCESS_STATEMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.3.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ProcessStatement object. All of the following methods are atomic.

15.3.3.1 Constructor Methods

The default constructor either initializes a process statement object from an unspecified source location, an unspecified declarator, no declarative items and no sequential statements or from a concurrent statement which has an equivalent process statement representation.

```
    IIR_ProcessStatement();
```

15.3.3.2 Postponed Methods

```
void
    set_postponed(        IR_Boolean postponed);
IR_Boolean
    get_postponed();
```

15.3.3.3 Destructor Method

The list destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
    ~IIR_ProcessStatement();
```

15.3.4 Predefined Public Data

The IIR_ProcessStatement's predefined public data includes the following data elements directly instantiated within the process statement:

```
IIR_DeclarationList              process_declarative_part;
IIR_SequentialStatementList      process_statement_part;
```

15.4 IIR_SensitizedProcessStatement

15.4.1 Derived Class Description

The predefined **IIR_SensitizedProcessStatement** class represents a process statment augmented by a single process sensitivity list. Such processes must appear within an architecture, concurrent block statement or concurrent generate statement. The sensitized process statement class, derived from the process statement class, adds an explicit sensitivity list.

15.4.2 Properties

TABLE 280. IIR_SensitizedProcessStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_SENSITIZED_PROCESS_STATEMENT**
Parent class	**IIR_ProcessStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.4.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SensitizedProcessStatement object. All of the following methods are atomic.

15.4.3.1 Constructor Method

The default constructor initializes a process statement object.

```
IIR_SensitizedProcessStatement();
```

15.4.3.2 Destructor Method

The list destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
    ~IIR_SensitizedProcessStatement();
```

15.4.4 Predefined Public Data

The IIR_SensitizedProcessStatement's predefined public data includes the following data elements directly instantiated within the process statement:

```
IIR_DesignatorList        sensitivity_list;
```

15.5 IIR_ConcurrentProcedureCallStatement

15.5.1 Derived Class Description

The predefined **IIR_ConcurrentProcedureCallStatement** represents a process containing a sequential procedure call statement and a wait statement.

15.5.2 Properties

TABLE 281. IIR_ConcurrentProcedureCallStatement Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONCURRENT_PROCEDURE_CALL_STATEMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.5.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConcurrentProcedureCallStatement object. All of the following methods are atomic.

15.5.3.1 Constructor Method

The default constructor initializes a concurrent procedure call statement object.

```
IIR_ConcurrentProcedurCallStatement();
```

15.5.3.2 Postponed Methods

```
void
    set_postponed(        IR_Boolean            postponed);
IR_Boolean
    get_postponed();
```

15.5.3.3 Subprogram Declaration Methods

```
void
    set_procedure_name(   IIR*                  procedure_name);
```

```
IIR*
    get_procedure_name();
```

15.5.3.4 Destructor Method

The list destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
    ~IIR_ConcurrentProcedureCallStatement();
```

15.5.4 Predefined Public Data

The IIR_ConcurrentProcedureCallStatement's predefined public data includes the following data elements directly instantiated within the concurrent procedure call statement:

```
IIR_AssociationList             actual_parameter_part;
IIR_SequentialStatementList     process_statement_part;
```

15.6 IIR_ConcurrentAssertionStatement

15.6.1 Derived Class Description

The predefined **IIR_ConcurrentAssertionStatement** represents a process containing a sequential assertion statement and a wait statement.

15.6.2 Properties

TABLE 282. IIR_ConcurrentAssertionStatementList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONCURRENT_ASSERTION_STATEMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Public data elements	None

15.6.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConcurrentAssertionStatement object. All of the following methods are atomic.

15.6.3.1 Constructor Method

The default constructor initializes a concurrent assertion statement object.

```
IIR_ConcurrentAssertionStatement();
```

15.6.3.2 Postponed Methods

```
void
    set_postponed(IR_Boolean      predicate);
IR_Boolean
    get_postponed();
```

15.6.3.3 Assertion Argument Methods

```
void
    set_assertion_condition(      IIR*           condition);
IIR*
```

```
    get_assertion_condition();
void
    set_report_expression(        IIR*           expression);
IIR*
    get_report_expression();
void
    set_severity_expression(      IIR*           expression);
IIR*
    get_severity_expression();
```

15.6.3.4 Destructor Method

The destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
    ~IIR_ConcurrentAssertionStatement();
```

15.7 IIR_ConcurrentConditionalSignalAssignment

15.7.1 Derived Class Description

The predefined **IIR_ConcurrentConditionalSignalAssignment** class represents a signal assignment wherein a nested if-then-else clause is evaluated to determine the waveform assigned to a target.

15.7.2 Properties

TABLE 283. IIR_ConcurrentStatementList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONCURRENT_CONDITIONAL_SIGNAL_ASSIGNMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.7.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConcurrentConditionalSignalAssignment object. All of the following methods are atomic.

15.7.3.1 Constructor Method

The default constructor initializes a concurrent conditional signal assignment object.

```
IIR_Concurrent ConditionalSignalAssignment();
```

15.7.3.2 Postponed Attribute Methods

```
void
    set_postponed( IR_Boolean            postponed);
IR_Boolean
    get_postponed();
```

15.7.3.3 Target Methods

```
void
    set_target(IIR*       t);
IIR*
    get_target();
```

15.7.3.4 Guarded Attribute Methods

```
void
    set_guarded(    IR_Boolean                guarded);
IR_Boolean
    get_guarded();
```

15.7.3.5 Delay Mechanism Methods

A signal assignment statement either uses transport or inertial delay.

```
void
    set_delay_mechanism(IIR_DelayMechanism       delay_mechanism);
IIR_DelayMechanism
    get_delay_mechanism();
```

15.7.3.6 Reject Time Methods

```
void
    set_reject_time_expression(    IIR*          reject_time_expression);
IIR*
    get_reject_time_expression();
```

15.7.3.7 Destructor Method

The list destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
    ~IIR_ConcurrentConditionalSignalAssignment();
```

15.7.4 Predefined Public Data

The IIR_ConcurrentConditionalSignalAssignment's predefined public data includes the following data elements directly instantiated within the assignment statement:

```
IIR_ConditionalWaveformList      conditional_waveforms
```

15.8 IIR_ConcurrentSelectedSignalAssignment

15.8.1 Derived Class Description

The predefined **IIR_ConcurrentSelectedSignalAssignment** class represents a signal assignment wherein an expression's value is compared against a list of choices to determine the waveform assigned to a target.

15.8.2 Properties

TABLE 284. IIR_ConcurrentStatementList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONCURRENT_SELECTED_SIGNAL_ASSIGNMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.8.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConcurrentSelectedSignalAssignment object. All of the following methods are atomic.

15.8.3.1 Constructor Method

The default constructor initializes a concurrent selected signal assignment object.

```
IIR_ConcurrentSelectedSignalAssignment();
```

15.8.3.2 Expression Methods

```
void
    set_expression(IIR*   expression);
IIR*
    get_expression();
```

15.8.3.3 Postponed Methods

```
void
    set_postponed( IR_Boolean           postponed);
IR_Boolean
    get_postponed();
```

15.8.3.4 Target Methods

```
void
    set_target(IIR*t);
IIR*
    get_target();
```

15.8.3.5 Guarded Attribute Methods

```
void
    set_guarded(   IR_Boolean     guarded);
IR_Boolean
    get_guarded();
```

15.8.3.6 Delay Mechanism Methods

A signal assignment statement either uses transport or inertial delay.

```
void
    set_delay_mechanism(   IIR_DelayMechanism     delay_mechanism);
IIR_DelayMechanism
    get_delay_mechanism();
```

15.8.3.7 Reject Time Methods

```
void
    set_reject_time_expression(   IIR*    reject_time_expression);
IIR*
    get_reject_time_expression();
```

15.8.3.8 Destructor Method

The list destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
    ~IIR_ConcurrentSelectedSignalAssignment();
```

15.8.4 Predefined Public Data

The IIR_ConcurrentSelectedSignalAssignment's predefined public data includes the following data elements directly instantiated within the assignment class:

```
IIR_SelectedWaveformList          selected_waveforms;
```

15.9 IIR_ComponentInstantiationStatement

15.9.1 Derived Class Description

The predefined **IIR_ComponentInstantiationStatement** represents the point at which an entity/architecture pair is inserted into the elaboration hierarchy.

15.9.2 Properties

TABLE 285. IIR_ConcurrentStatementList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_COMPONENT_INSTANTIATION_STATEMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.9.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ComponentInstantiationStatement object. All of the following methods are atomic.

15.9.3.1 Constructor Method

The default constructor initializes a component instantiation statement object.

```
IIR_ComponentInstantiationStatement();
```

15.9.3.2 Instantiated Unit Methods

```
void
    set_instantiated_unit( IIR*            instantiated_unit);
IIR*
    get_instantiated_unit();
```

15.9.3.3 Destructor Method

The list destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
   ~IIR_ComponentInstantiationStatement();
```

15.9.4 Predefined Public Data

The IIR_ComponentInstantiationStatement's predefined public data includes the following data elements directly instantiated within the component instantiation statement:

```
IIR_AssociationList            generic_map_aspect;
IIR_AssociationList            port_map_aspect;
```

15.10 IIR_ConcurrentGenerateForStatement

15.10.1 Derived Class Description

The predefined **IIR_ConcurrentGenerateForStatement** class represents a block statement which is elaborated zero or more times depending on the evaluation of a discrete range during elaboration.

15.10.2 Properties

TABLE 286. IIR_ConcurrentStatementList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONCURRENT_GENERATE_FOR_STATEMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.10.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConcurrentGenerateForStatement object. All of the following methods are atomic.

15.10.3.1 Constructor Method

The default constructor initializes a concurrent generate for statement object from an unspecified source location, specified declarator, NIL constant declaration representing the generate parameter specification, no block declarative items and no block statements

```
IIR_ConcurrentGenerateForStatement();
```

15.10.3.2 For Generate Scheme Methods

```
void
    set_generate_parameter_specification(IIR_ConstantDeclaration*
                                    generate_parameter_specification);

IIR_ConstantDeclaration*
    get_generate_parameter_specification();
```

15.10.3.3 Destructor Method

The list destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
    ~IIR_ConcurrentGenerateForStatement();
```

15.10.4 Predefined Public Data

The IIR_ConcurrentGenerateForStatement's predefined public data includes the following data elements directly instantiated within the block statement:

```
IIR_DeclarationList              block_declarative_part;
IIR_ConcurrentStatementList      concurrent_statement_part;
```

15.11 IIR_ConcurrentGenerateIfStatement

15.11.1 Derived Class Description

The predefined **IIR_ConcurrentGenerateIfStatement** represents a block which is either elaborated once or not at all, depending on the value of a boolean condition.

15.11.2 Properties

TABLE 287. IIR_ConcurrentStatementList Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*
IR_Kind enumeration value	**IR_CONCURRENT_GENERATE_IF_STATEMENT**
Parent class	**IIR_ConcurrentStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

15.11.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_BlockGenerateIfStatement object. All of the following methods are atomic.

15.11.3.1 Constructor Method

The default constructor initializes a concurrent generate if statement object from the specified source location, specified declarator, NIL if condition, no block declarative items and no block statements.

```
    IIR_ConcurrentGenerateIfStatement();
```

15.11.3.2 If Condition Methods

```
void
    set_if_condition(      IIR*                     condition);
IIR*
    get_if_condition();
```

15.11.3.3 Destructor Method

The list destructor method deletes the label, guard expression and all lists associated with the block statement, then deletes the block statement itself. Note that in the case of canonical objects, such as identifiers, other references to the canonical object may prevent the IIR implementation from actually deleting the label.

```
void
    ~IIR_ConcurrentGenerateIfStatement();
```

15.11.4 Predefined Public Data

The IIR_ConcurrentGenerateIfStatement's predefined public data includes the following data elements directly instantiated within the concurrent generate statement:

```
IIR_DeclarationList            block_declarative_part;
IIR_ConcurrentStatementList    concurrent_statement_part;
```

CHAPTER 16

IIR_SimultaneousStatement Derived Classes

This chapter specifies the properties, predefined public methods and predefined public data associated with the IIR_SimultaneousStatement class and all predefined classes derived from IIR_SimultaneousStatement (as shown in Table 295 on page 534). All derivative classes of IIR_SimultaneousStatement are dynamically and individually allocated.

Application-specific data elements may be added to an optional extension class layer just after differentiating the different kinds of concurrent statements. For example, these extensions may provide additional debugging information or information specific to a particular backend. Application-specific methods may be added to extension layers anywhere in the IIR class hierarchy. New application-specific classes may be created from either the IIR_SimultaneousStatement or one of its derivatives.

TABLE 288. Class hierarchy derived from IIR_SimultaneousStatement

	Level 4 Derived Classes (only classes derived from IIR_SimultaneousStatement)		Level 5 Derived Classes		Level 6 Derived Classes
E	*IIR_SimpleSimultaneousStatement*				
E	*IIR_ConcurrentBreakStatement*				
E	*IIR_SimultaneousIfStatement*				
E	*IIR_SimultaneousCaseStatement*				
E	*IIR_SimultaneousProceduralStatement*				
E	*IIR_SimultaneousNullStatement*				

16.1 IIR_SimultaneousStatement

16.1.1 Derived Class Description

The simultaneous statement derived classes are all derived via the **IIR_SimultaneousStatement** class.

16.1.2 Properties

TABLE 289. IIR_SimultaneousStatement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	None, not directly instantiated
Parent class	**IIR_Statement**
Predefined child classes	**IIR_SimpleSimultaneousStatement** **IIR_ConcurrentBreakStatement** **IIR_SimultaneousIfStatement** **IIR_SimultaneousCaseStatement** **IIR_SimultaneousProceduralStatement** **IIR_SimultaneousNullStatement**
Instantiation?	Indirectly via any of the derived classes of IIR_SimultaneousStatement
Application-specific data elements	Via extension classes associated with specific derived classes of the IIR_SimultaneousStatement class
Public data elements	None

16.1.3 Predefined Public Methods

All of the following methods must be applied to a valid object having a class derived from IIR_SimultaneousStatement. All of the following methods are atomic.

16.2 IIR_SimpleSimultaneousStatement

16.2.1 Derived Class Description

The predefined IIR_SimpleSimultaneousStatement class describes zero or more characteristic expressions.

16.2.2 Properties

TABLE 290. IIR_SimpleSimultaneousStatement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMPLE_SIMULTANEOUS_STATEMENT**
Parent class	**IIR_SimultaneousStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Predefined public data elements	None

16.2.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimpleSimultaneousStatement object. All of the following methods are atomic.

16.2.3.1 Constructor Methods

The constructors initialize a simple simultaneous statement object from an unspecified source location, an unspecified declarator, an undefined left hand expression and an undefined right hand expression.

```
IIR_SimpleSimultaneousStatement();
```

16.2.3.2 Left Hand Expression Methods

```
void
    set_left_expression(IIR*      left_expression);
IIR*
    get_left_expression();
```

16.2.3.3 Right Hand Expression Methods

```
void
    set_right_expression(IIR*     right_expression);
IIR*
    get_right_expression();
```

16.2.3.4 Tolerence Methods

```
void
    set_tolerence_aspect(IIR*     tolerence_aspect);
IIR*
    get_tolerence_aspect();
```

16.2.3.5 Destructor Method

```
void
    ~IIR_SimpleSimultaneousStatement();
```

16.3 IIR_ConcurrentBreakStatement

16.3.1 Derived Class Description

The predefined IIR_ConcurrentBreakStatement class .

16.3.2 Properties

TABLE 291. IIR_ConcurrentBreakStatement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_CONCURRENT_BREAK_STATEMENT**
Parent class	**IIR_SimultaneousStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

16.3.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_ConcurrentBreakStatement object. All of the following methods are atomic.

16.3.3.1 Constructor Methods

```
    IIR_ConcurrentBreakStatement();
```

16.3.3.2 Condition Methods

```
void
    set_condition(IIR*     condition);
IIR*
    get_condition();
```

16.3.3.3 Destructor Method

```
void
    ~IIR_ConcurrentBreakStatement();
```

16.3.4 Predefined Public Data

```
IIR_BreakList             break_list;
IIR_DesignatorList        sensitivity_clause;
```

16.4 IIR_SimultaneousIfStatement

16.4.1 Derived Class Description

The predefined IIR_SimultaneousIfStatement class uses a nested sequence of boolean expressions to choose among simultaneous statement parts.

16.4.2 Properties

TABLE 292. IIR_SimultaneousIfStatement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMULTANEOUS_IF_STATEMENT**
Parent class	**IIR_SimultaneousStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

16.4.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousIfStatement object. All of the following methods are atomic.

16.4.3.1 Constructor Methods

The constructors initialize a block statement object from an unspecified source location, an unspecified declarator, no generic clause items, no generic map items, no port clause items, no port map items, no block declarative items and no block statements.

```
IIR_SimultaneousIfStatement();
```

16.4.3.2 Condition Methods

```
void
    set_condition(IIR*     condition);
IIR*
    get_condition();
```

16.4.3.3 Simultaneous Elsif Methods

```
void
    set_elsif(IIR_SimultaneousElsif*            elsif);
IIR_SimultaneousElsif
    get_elsif();
```

16.4.3.4 Destructor Method

```
void
    ~IIR_SimultaneousIfStatement();
```

16.4.4 Predefined Public Data Elements

```
IIR_SimultaneousStatementList            then_statement_list;
IIR_SimultaneousStatementList            else_statement_list;
```

16.5 IIR_SimultaneousCaseStatement

16.5.1 Derived Class Description

The predefined IIR_SimultaneousCaseStatement class uses an expression of discrete type to select an unique simultaneous statement part for execution.

16.5.2 Properties

TABLE 293. IIR_SimultaneousCaseStatement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMULTANEOUS_CASE_STATEMENT**
Parent class	**IIR_SimultaneousStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

16.5.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousCaseStatement object. All of the following methods are atomic.

16.5.3.1 Constructor Methods

The constructors initialize a simultaneous case statement object from an unspecified source location, an unspecified declarator, an undefined condition and no simultaneous alternatives.

```
    IIR_SimultaneousCaseStatement();
```

16.5.3.2 Expression Methods

```
void
    set_expression(IIR*   expression);
IIR*
    get_expression();
```

16.5.3.3 Destructor Method

```
void
    ~IIR_SimultaneousCaseStatement();
```

16.5.4 Predefined Public Data

```
IIR_SimultaneousAlternativeList          simultaneous_alternative_list;
```

16.6 IIR_SimultaneousProceduralStatement

16.6.1 Derived Class Description

The predefined **IIR_SimultaneousProceduralStatement** class represents differential and algebraic equations using a sequential notation.

16.6.2 Properties

TABLE 294. IIR_SimultaneousProceduralStatement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMULTANEOUS_PROCEDURAL_STATEMENT**
Parent class	**IIR_SimultaneousStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class
Predefined public data elements	None

16.6.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousProceduralStatement object. All of the following methods are atomic.

16.6.3.1 Constructor Methods

The constructors initialize a block statement object from an unspecified source location, an unspecified declarator, .

```
IIR_SimultaneousProceduralStatement();
```

16.6.3.2 Destructor Method

```
void
    ~IIR_SimultaneousProceduralStatement();
```

16.6.4 Predefined Public Data Elements

```
IIR_DeclarationList                  procedural_declarative_part;
IIR_SequentialStatementList          procedural_statement_part;
```

16.7 IIR_SimultaneousNullStatement

16.7.1 Derived Class Description

The predefined **IIR_SimultaneousNullStatement** class acts as a statement place-holder, generating no characteristic expressions.

16.7.2 Properties

TABLE 295. IIR_SimultaneousNullStatement Properties

Applicable language(s)	VHDL-AMS
IR_Kind enumeration value	**IR_SIMULTANEOUS_NULL_STATEMENT**
Parent class	**IIR_SimultaneousStatement**
Predefined child classes	None
Instantiation?	Dynamically via new
Application-specific data elements	Via extension class

16.7.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_SimultaneousNullStatement object. All of the following methods are atomic.

16.7.3.1 Constructor Methods

The constructors initialize a simultaneous null statement object from an unspecified source location and an unspecified label.

```
IIR_SimultaneousNullStatement();
```

16.7.3.2 Destructor Method

```
void
    ~IIR_SimultaneousNullStatement();
```

CHAPTER 17 *Post-Elaboration*

17.1 Overview

The VHDL language reference manual defines both a static and a dynamic elaboration process.

17.2 Post-Elaboration Classes

Classes shown in Table 296 on page 536 through Table 301 on page 541 may occur in a post-elaboration data-

TABLE 296. File-Related and Literal Classes Appearing Post-Elaboration

File-Related Classes Appearing Post-Elaboration	Literal Classes Appearing Post-Elaboration
IR_DESIGN_FILE	IR_IDENTIFIER
IR_COMMENT	IR_CHARACTER_LITERAL
	IR_STRING_LITERAL
	IR_BIT_STRING_LITERAL
	IR_INTEGER_LITERAL
	IR_INTEGER_LITERAL32
	IR_INTEGER_LITERAL64
	IR_FLOATING_POINT_LITERAL
	IR_FLOATING_POINT_LITERAL32
	IR_FLOATING_POINT_LITERAL64

base.

TABLE 297. Tuple and List Classes Appearing Post-Elaboration

Tuple Classes Appearing Post-Elaboration	List Classes Appearing Post-Elaboration
IR_ASSOCIATION_ELEMENT_BY_EXPRESSION	IR_ASSOCIATION_LIST
IR_ASSOCIATION_ELEMENT_BY_OTHERS	IR_ATTRIBUTE_SPECIFICATION_LIST
IR_ASSOCIATION_ELEMENT_OPEN	IR_BREAK_LIST
IR_BREAK_ELEMENT	IR_CASE_ALTERNATIVE_LIST
IR_CASE_STATEMENT_ALTERNATIVE_BY_EXPRESSION	IR_CHOICE_LIST
IR_CASE_STATEMENT_ALTERNATIVE_BY_CHOICES	IR_COMMENT_LIST
IR_CASE_STATEMENT_ALTERNATIVE_BY_OTHERS	IR_CONCURRENT_STATEMENT_LIST
IR_CHOICE	IR_CONDITIONAL_WAVEFORM_LIST
IR_CONDITIONAL_WAVEFORM	IR_CONFIGURATION_ITEM_LIST
IR_COMPONENT_SPECIFICATION	IR_DECLARATION_LIST
IR_BLOCK_CONFIGURATION	IR_DESIGN_FILE_LIST
IR_COMPONENT_CONFIGURATION	IR_DESIGNATOR_LIST
IR_DESIGNATOR_EXPLICIT	IR_ELEMENT_DECLARATION_LIST
IR_DESIGNATOR_BY_OTHERS	IR_NATURE_ELEMENT_DECLARATION_LIST
IR_DESIGNATOR_BY_ALL	IR_ENTITY_CLASS_ENTRY_LIST
IR_ELSIF	IR_ENUMERATION_LITERAL_LIST
IR_ENTITY_CLASS_ENTRY	IR_GENERIC_LIST
IR_SELECTED_WAVEFORM	IR_INTERFACE_LIST
IR_SIMULTANEOUS_ALTERNATIVE_BY_EXPRESSION	IR_LIBRARY_UNIT_LIST
IR_SIMULTANEOUS_ALTERNATIVE_BY_CHOICES	IR_PORT_LIST
IR_SIMULTANEOUS_ALTERNATIVE_BY_OTHERS	IR_SELECTED_WAVEFORM_LIST
IR_SIMULTANEOUS_ELSIF	IR_SEQUENTIAL_STATEMENT_LIST
IR_WAVEFORM_ELEMENT	IR_SIMULTANEOUS_ALTERNATIVE_LIST
	IR_SIMULTANEOUS_STATEMENT_LIST
	IR_STATEMENT_LIST
	IR_UNIT_LIST
	IR_WAVEFORM_LIST

TABLE 298. Type, Nature and Declaration Classes Appearing Post-Elaboration

Type & Nature Classes Appearing Post-Elaboration	Declaration Classes Appearing Post-Elaboration
IR_ENUMERATION_TYPE_DEFINITION	IR_FUNCTION_DECLARATION
IR_ENUMERATION_SUBTYPE_DEFINITION	IR_PROCEDURE_DECLARATION
IR_INTEGER_TYPE_DEFINITION	IR_ELEMENT_DECLARATION
IR_INTEGER_SUBTYPE_DEFINITION	IR_NATURE_ELEMENT_DECLARATION
IR_FLOATING_TYPE_DEFINITION	IR_ENUMERATION_LITERAL
IR_FLOATING_SUBTYPE_DEFINITION	IR_TYPE_DECLARATION
IR_PHYSICAL_TYPE_DEFINITION	IR_SUBTYPE_DECLARATION
IR_PHYSICAL_SUBTYPE_DEFINITION	IR_NATURE_DECLARATION
IR_RANGE_TYPE_DEFINITION	IR_SUBNATURE_DECLARATION
IR_SCALAR_NATURE_DEFINITION	IR_CONSTANT_DECLARATION
IR_SCALAR_SUBNATURE_DEFINITION	IR_FILE_DECLARATION
IR_ARRAY_TYPE_DEFINITION	IR_SIGNAL_DECLARATION
IR_ARRAY_SUBTYPE_DEFINITION	IR_SHARED_VARIABLE_DECLARATION
IR_ARRAY_NATURE_DEFINITION	IR_VARIABLE_DECLARATION
IR_ARRAY_SUBNATURE_DEFINITION	IR_TERMINAL_DECLARATION
IR_RECORD_TYPE_DEFINITION	IR_FREE_QUANTITY_DECLARATION
IR_RECORD_SUBTYPE_DEFINITION	IR_ACROSS_QUANTITY_DECLARATION
IR_RECORD_NATURE_DEFINITION	IR_THROUGH_QUANTITY_DECLARATION
IR_RECORD_SUBNATURE_DEFINITION	IR_SPECTRUM_SOURCE_QUANTITY_DECLARATION
IR_ACCESS_TYPE_DEFINITION	IR_NOISE_SOURCE_QUANTITY_DECLARATION
IR_ACCESS_SUBTYPE_DEFINITION	IR_CONSTANT_INTERFACE_DECLARATION
IR_FILE_TYPE_DEFINITION	IR_FILE_INTERFACE_DECLARATION
IR_SIGNATURE	IR_SIGNAL_INTERFACE_DECLARATION
	IR_VARIABLE_INTERFACE_DECLARATION
	IR_TERMINAL_INTERFACE_DECLARATION
	IR_QUANTITY_INTERFACE_DECLARATION
	IR_ALIAS_DECLARATION
	IR_ATTRIBUTE_DECLARATION
	IR_COMPONENT_DECLARATION
	IR_GROUP_DECLARATION
	IR_GROUP_TEMPLATE_DECLARATION
	IR_LIBRARY_DECLARATION
	IR_ENTITY_DECLARATION
	IR_ARCHITECTURE_DECLARATION
	IR_PACKAGE_DECLARATION
	IR_PACKAGE_BODY_DECLARATION
	IR_CONFIGURATION_DECLARATION
	IR_PHYSICAL_UNIT
	IR_ATTRIBUTE_SPECIFICATION
	IR_CONFIGURATION_SPECIFICATION
	IR_DISCONNECTION_SPECIFICATION
	IR_LABEL
	IR_LIBRARY_CLAUSE
	IR_USE_CLAUSE

TABLE 299. name and Operator Classes Appearing Post-Elaboration

Name Classes Appearing Post-Elaboration	Operator Classes Appearing Post-Elaboration
IR_SIMPLE_NAME	IR_IDENTITY_OPERATOR
IR_SELECTED_NAME	IR_NEGATION_OPERATOR
IR_SELECTED_NAME_BY_ALL	IR_ABSOLUTE_OPERATOR
IR_INDEXED_NAME	IR_NOT_OPERATOR
IR_SLICE_NAME	IR_AND_OPERATOR
IR_USER_ATTRIBUTE	IR_OR_OPERATOR
IR_BASE_ATTRIBUTE	IR_NAND_OPERATOR
IR_LEFT_ATTRIBUTE	IR_NOR_OPERATOR
IR_RIGHT_ATTRIBUTE	IR_XOR_OPERATOR
IR_LOW_ATTRIBUTE	IR_XNOR_OPERATOR
IR_HIGH_ATTRIBUTE	IR_EQUALITY_OPERATOR
IR_ASCENDING_ATTRIBUTE	IR_INEQUALITY_OPERATOR
IR_IMAGE_ATTRIBUTE	IR_LESS_THAN_OPERATOR
IR_VALUE_ATTRIBUTE	IR_LESS_THAN_OR_EQUAL_OPERATOR
IR_POS_ATTRIBUTE	IR_GREATER_THAN_OPERATOR
IR_VAL_ATTRIBUTE	IR_GREATER_THAN_OR_EQUAL_OPERATOR
IR_SUCC_ATTRIBUTE	IR_SLL_OPERATOR
IR_PRED_ATTRIBUTE	IR_SRL_OPERATOR
IR_LEFT_OF_ATTRIBUTE	IR_SLA_OPERATOR
IR_RIGHT_OF_ATTRIBUTE	IR_SRA_OPERATOR
IR_DELAYED_ATTRIBUTE	IR_ROL_OPERATOR
IR_STABLE_ATTRIBUTE	IR_ROR_OPERATOR
IR_QUIET_ATTRIBUTE	IR_ADDITION_OPERATOR
IR_TRANSACTION_ATTRIBUTE	IR_SUBTRACTION_OPERATOR
IR_EVENT_ATTRIBUTE	IR_CONCATENATION_OPERATOR
IR_ACTIVE_ATTRIBUTE	IR_MULTIPLICATION_OPERATOR
IR_LAST_EVENT_ATTRIBUTE	IR_DIVISION_OPERATOR
IR_LAST_ACTIVE_ATTRIBUTE	IR_MODULUS_OPERATOR
IR_LAST_VALUE_ATTRIBUTE	IR_REMAINDER_OPERATOR
IR_BEHAVIOR_ATTRIBUTE	IR_EXPONENTIATION_OPERATOR
IR_STRUCTURE_ATTRIBUTE	IR_FUNCTION_CALL
IR_DRIVING_ATTRIBUTE	IR_PHYSICAL_LITERAL
IR_DRIVING_VALUE_ATTRIBUTE	IR_AGGREGATE
IR_PATH_NAME_ATTRIBUTE	IR_OTHERS_INITIALIZATION
IR_ACROSS_ATTRIBUTE	IR_QUALIFIED_EXPRESSION
IR_THROUGH_ATTRIBUTE	IR_TYPE_CONVERSION
IR_REFERENCE_ATTRIBUTE	IR_ALLOCATOR
IR_CONTRIBUTION_ATTRIBUTE	
IR_TOLERANCE_ATTRIBUTE	
IR_DOT_ATTRIBUTE	
IR_INTEG_ATTRIBUTE	
IR_ABOVE_ATTRIBUTE	
IR_ZOH_ATTRIBUTE	
IR_LTF_ATTRIBUTE	
IR_ZTF_ATTRIBUTE	
IR_RAMP_ATTRIBUTE	
IR_SLEW_ATTRIBUTE	

TABLE 300. Sequential and Concurrent Classes Appearing Post-Elaboration

Sequential Statement Classes Appearing Post-Elaboration	Concurrent Statement Classes Appearing Post-Elaboration
IR_WAIT_STATEMENT	IR_PROCESS_STATEMENT
IR_ASSERTION_STATEMENT	IR_SENSITIZED_PROCESS_STATEMENT
IR_REPORT_STATEMENT	IR_CONCURRENT_PROCEDURE_CALL_STATEMENT
IR_SIGNAL_ASSIGNMENT_STATEMENT	IR_CONCURRENT_ASSERTION_STATEMENT
IR_VARIABLE_ASSIGNMENT_STATEMENT	IR_CONCURRENT_CONDITIONAL_SIGNAL_ASSIGNMENT
IR_PROCEDURE_CALL_STATEMENT	IR_CONCURRENT_SELECTED_SIGNAL_ASSIGNMENT
IR_IF_STATEMENT	IR_CONCURRENT_INSTANTIATION_STATEMENT
IR_CASE_STATEMENT	IR_CONCURRENT_GENERATE_FOR_STATEMENT
IR_FOR_LOOP_STATEMENT	IR_CONCURRENT_GENERATE_IF_STATEMENT
IR_WHILE_LOOP_STATEMENT	
IR_NEXT_STATEMENT	
IR_EXIT_STATEMENT	
IR_RETURN_STATEMENT	
IR_NULL_STATEMENT	
IR_BREAK_STATEMENT	

TABLE 301. Simultaneous and Elaboration-only Classes Appearing Post-Elaboration

Simultaneous Statement Classes Appearing Post-Elaboration	Elaboration-only Classes Appearing Post-Elaboration
IR_SIMPLE_SIMULTANEOUS_STATEMENT	IR_DRIVER
IR_CONCURRENT_BREAK_STATEMENT	IR_EFFECTIVE_VALUE;
IR_SIMULTANEOUS_IF_STATEMENT	
IR_SIMULTANEOUS_CASE_STATEMENT	
IR_SIMULTANEOUS_PROCEDURAL_STATEMENT	
IR_SIMULTANEOUS_NULL_STATEMENT	

17.3 IIR_Driver

17.3.1 Derived Class Description

The predefined **IIR_Driver** classes represent the projected waveform elements resulting from signal assignment statements within a single process.

17.3.2 Properties

TABLE 302. IIR_Driver Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_DRIVER**
Parent class	**IIR**
Predefined child classes	**NONE**
Instantiation?	Dynamically via new
Application-specific data elements	Via extension classes
Predefined public data elements	None

17.3.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_Driver object. All of the following methods are atomic.

17.3.3.1 Constructor Method

The constructor initializes an effective value object with an undefined source location, an undefined actual, and undefined next value.

```
    IIR_Driver();
```

17.3.4 Signal Methods

```
void
    set_signal(IIR* signal);
IIR*
    get_signal();
```

17.3.4.2 Value Methods

```
void
```

```
    set_signal(IIR* signal);
IIR*
    get_signal();
```

17.3.4.3 Destructor Method

```
    ~IIR_Driver();
```

17.4 IIR_EffectiveValue

17.4.1 Derived Class Description

The predefined **IIR_EffectiveValue** class represents the value of a signal from within a specific process.

17.4.2 Properties

TABLE 303. IIR_EffectiveValue Properties

Applicable language(s)	VHDL-87, VHDL-93, VHDL-98*,VHDL-AMS
IR_Kind enumeration value	**IR_DRIVER**
Parent class	**IIR**
Predefined child classes	**NONE**
Instantiation?	Dynamically via new
Application-specific data elements	Via Extension Classes
Predefined public data elements	None

17.4.3 Predefined Public Methods

Except for the constructor, all of the following methods must be applied to a valid IIR_EffectiveValue object. All of the following methods are atomic.

17.4.3.1 Constructor Method

The constructor initializes an effective value object with an undefined source location, an undefined actual, and undefined next value.

```
IIR_EffectiveValue();
```

17.4.4 Signal Methods

```
void
    set_signal(IIR* signal);
IIR*
    get_signal();
```

17.4.4.2 Value Methods

```
void
    set_signal(IIR* signal);
```

```
IIR*
    get_signal();
```

17.4.4.3 Destructor Method

```
    ~IIR_EffectiveValue();
```

CHAPTER 18

Implementation of AIRE/CE in C++, C, Ada and Eiffel

This chapter is still in development.

CHAPTER 19

Version Change Log

This chapter represents the changes from one version to another since the specification was placed under revision change control. Changes from the latest version to the earliest are found in reverse chronological order. Each revision set also includes the status of revision approvals by the AIRE Change Review Board.

The document was first placed under change control with Version 2.3 (released 4/16/96). This trial implementation draft was the first complete enough to begin implementation of IIR foundation implementations.

This section is intended to meet ISO-9000 process control requirements.

19.1 Version 2.3 to 2.4

An IIR_List entry was defined, as implied by the class overview specifications. This class inherited list length methods from all of the classes derived from IIR_List.

Within the IIR class name-space, methods and data elements which are not predefined must be prefaced with an '_' *except for constructors, destructors and operators.*

Source location information was made accessible via base class methods.

Attribute specification lists and disconnect specification lists were merged into declaration lists.

Binding indications and component specifications were merged into component configurations and configuration specifications (extension classes with data elements would have broken binary IIR implementations).

Identifier lists were eliminated (they broke cannonization).

19.2 Version 2.4 to 2.5

IIR_DesignUnitList was changed to IIR_LibraryUnitList in order to bring AIRE intro correspondence with the IIR_LibraryUnit and the VHDL LRM.

The IIR_Kind definition incorrectly listed IIR_SELECT_WAVEFORM rather than IIR_SELECTED_WAVEFORM.

The IIR_Kind definition incorrectly listed IIR_UNIT_DECLARATION rather than IIR_PHYSICAL_UNIT.

The IIR_DesignFile methods for setting and getting the file name are redundent with the base class's method and were removed.

For scalar type definitions, methods and parameters were ordered (left, direction, right) in correspondence to the VHDL langauge design.

Lists of signals, such as those which appear in VHDL wait statements and process sensitivity lists, are now represented by IIR_SignalName and IIR_SignalNameList. The IIR_NameList node was removed.

The string "Literal" was added to all literal classes and the string "_LITERAL" was added to the enumeration labels associated with all literals.

There were numerous corrections to type signatures and formatting.

Hypertext linkages were added to chapters on basic data types, names, expressions, sequential statements and concurrent statements.

Hypertext linkages still need to be removed from each table entry (redundent with inline text). GIF images are needed for the line-art.

19.3 Version 2.5 to 2.6

IIR_EntityNameList and IIR_ExpressionList classes are not referenced and were deleted.

Since IIR_InstantiationList is directly instantiated in component configurations and configuration specifications, it made no sense to differentiate three derived classes. Instead a new tuple was created, IIR_InstantiationElement with three subclasses: IIR_InstantiationElementByName, IIR_InstantiationElementByOthers and IIR_InstantiationElementByAll.

Acquisition of an IIR_CharacterLiteral changed from a constructor to get method, consistent with the possible cannonization of other literals.

Representation of IIR_DisconnectionSpecifications changed for consistency (variously listed as IIR_DisconnectSpecification and IIR_DisconnectionSpecification) and internal representation (consistency and to use the new IIR_SignalNameList class.

All next methods were made implementation-dependent (not part of AIRE) since they were subsumed in list object methods.

Dyadic expression left and right operand methods were corrected to read set_...() and get_...().

Association of architectures to entities and configurations to entities now done directly with a set_...() method rather than an associate method (change made for consistency).

19.4 Version 2.6 to 2.7

IIR_Kind was changed to IR_Kind in order to emphasize commonality between the IIR and FIR

Name lookup methods changed to return 0 or more IIR_Declarations.

The IIR_AttributeSpecificationList was restored.

An IIR_Elsif tuple was added to represent nested else clauses.

19.5 Version 2.7 to 2.8

VHDL attribute specifications associate a value with one or more named entities. In order to explicitly a specific subprogram as a named entity, a type signature is required. Since IIR_DesignatorExplicit denotes an explicitly named entity, we proposed to add two methods to IIR_DesignatorExplicit:

```
void
    set_signature(IIR_Signature*        signature);
IIR_Signature*
   get_signature();
```

VHDL's If statement provides for a nested set of if-then-elsif-else clauses. The IIR_IfStatement currently provides for an if condition (a pointer), an elsif tuple (a pointer which can be recursively nested and a then statement list. Rather than allocate an IIR_Elsif tuple to simple if-then-else statements, we propose to an a new public data element to the IIR_IfStatement:

```
IIR_SequentialStatementList                else_sequence;
```

VHDL's array types include arrays with both constrained and unconstrained array indicies. Whereas an array type cannot be instantiated until a constraint has been imposed on the type, the IIR must represent such an unconstrained index within type and interface declarations. The index is indirectly constrained in so far as the eventually constraint must lie within the domain of the subtype denoted by the type mark preceeding the

RANGE <> notation.

The suggested representation (following VHDL semantics) is to utilize an index subtype (some IIR_<something>SubtypeDefinition) where the left and right limits are NIL pointers, however the subtype's base type is the one denoted by the type mark preceeding the RANGE <> notation.

This points to another note about the IIR. A subtype can have another base type which is itself a subtype. For example, natural is a subtype of integer; one can create your own subtype of natural. Thus the base type references may in fact refer to another subtype.

Multi-dimensional arrays and arrays include record (or other elements) may be represented using composite array elements. For example, the first dimension of a two-dimensional array would have an element which is itself an array.

Since arrays may include elements of access type, the methods must be corrected as follows:

```
void
    set_element_subtype(IIR_TypeDefinition*        subtype);
IIR_TypeDefinition*
    get_element_subtype();
```

where the subtype is either a scalar, access or file type (not composite).

Signatures are the on explicit list of types in VHDL. Since signatures are not a very common use of type marks, an IIR_DesignatorList replaces the IIR_TypeDefinitionList previously found within the public data elements of a signature. All references to IIR_TypeDefinition should be deleted.

The proposal is to use constructors and destructors for all instantiable classes, but to retain notations that the class can potentially be made cannonical by a compliant implementation. Serafin notes that constructors improve the definition and implementation.

Most of the primitive types can be shared between the IIR and FIR, thus the names should be changed accordingly for:

1. IR_Boolean
2. IR_Char
3. IR_Int32
4. IR_Int64
5. IR_FP32
6. IR_FP64
7. IR_Kind
8. IR_SignalKind
9. IR_Mode
10. IR_Pure
11. IR_DelayMechanism
12. (but not IIR pointer since it is IIR specific)

When writing "wrappers" within the (SAVANT) symbol table code, Dale notes that the IIR_StringLiteral and IIR_Identifier use methods which are very closely related in form. Unfortunately, methods for accessing the value and length of string literals and text are slightly different. In order to give humans less opportunity for confusion, it is suggested that IIR_Identifier methods be changed as follows:

get_value, set_value, get_length and set_length

(mirroring the literal method names).

In Section 12.14 (File Declarations) the names in this section should be changed to get_file_logical_name and set_file_logical_name (cut/ paste error).

IIR_PhysicalTypeDefinition should refer to IIR_PhysicalUnit (not IIR_UnitDeclaration). Note that we can change the name of the physical unit to unit declaration, however they need to be the same functional class.

Resolution functions are missing from all six of the subtype definitions: (enumeration, integer, floating, physical, array, and access). Furthermore, a record subtype definition is required specifically in order to add resolution functions. The suggested form is:

```
void
    set_resolution_function(IIR_FunctionDeclaration*    resolution_function);
IIR_FunctionDeclaration*
   get_resolution_function();
```

A resolution function argument should be added to all of the subtype get methods as the last argument, perhaps defaulting to NIL.

In the IIR_GroupDeclaration definition, the public data element denoting the list of group constituents must be an IIR_DesignatorList denoting a collection of named entities (IIR_GroupConstituentLists don't exist and have the same semantic content as IIR_DesignatorLists).

The VHDL allocator expression can either take the form of a subtype indication (already provided for by the IIR) or a qualified expression (not provided for).

A suggested solution is to add initializer methods to the IIR_Allocator:

```
void
    set_value(IIR*          value);
IIR*
   get_value();
```

Where the value is either a qualified expression or the default value of the subtype indication (second is optional since it can be constructed on-the-fly from the subtype definition).

The parameter denoting the dimension is missing from several attributes:

IIR_HighAttribute

IIR_LowAttribute

IIR_ReverseRangeAttribute

In all cases this suffix is a single (optional) pointer to IIR. Note that we could replace 'suffix' with 'parameter' if this would be more consistent with the LRM. Thus the proposal is to add:

```
void
    set_suffix(IIR*         suffix);
IIR*
    get_suffix();
```

(When we make this correction I'll try to catch any other missing suffix).

Table 14.7.2 is missing IIR_SLAOperator. This should be added.

The InstantiationList has been replaced by the IIR_DesignatorList and should be deleted.

19.6 Version 2.8 to Version 3.0

Several folks found need for a LibraryDeclaration containing zero or more primary design units. Note that this is distinct from a library clause (which brings a library into visibility).

Tim McBrayer notes that unless an entity_class is added to the IIR_Designator, it may be impossible to completely reconstruct the source VHDL.

The component configuration's internal block configuration was changed from being a static data element (which did not make sense) to set and get methods (Thanks to Dale Martin).

Literal class names in the literal class table were updated for consistency with the remainder of the document (Thanks to Rob Newshutz).

The IIR_EntityClassEntry's representation was changed to refer to IR_Kind and a IR_Boolean designating boxed (potentially repeated) entries. This narrowed the typing and provided a means to represent boxed entries.

The IIR_SignalName and IIR_GroupConstituent tuples were both functionally replaced by the IIR_Designator, and thus were removed from the specification.

A third alternative was added to **IIR_CaseStatementAlternative**: **IIR_CaseStatementAlternativeByChoices** in order to denote a single sequential statement list applied to two or more choices. The **IIR_CaseStatementAlternativeByChoices** contains an **IIR_ChoiceList** (also new). The **IIR_ChoiceList**s consists of two or more, independently allocated **IIR_Choice**. (Thanks to Tim McBrayer for this optimization).

The expression was missing from the concurrent selected signal assignment statement (Thanks to Malolan Chetlur).

The following new classes were added to support VHDL-AMS (Dave Barton's draft LRM of Mid-June, 1996):

- IIR_SimultaneousElsif, IIR_SimultaneousAlternative, IIR_BreakElement (all found in chapter on tuples)
- IIR_SimultaneousStatementList (found in chapter on lists)
- IIR_BreakList (found in chapter on lists)
- IIR_SimultaneousAlternativeList (found in chapter on lists)
- IIR_ScalarNatureDefinition (found in chapter on types)
- IIR_CompositeNatureDefinition with subclasses IIR_ArrayNatureDefinition and IIR_RecordNatureDefinition (all found in chapter on types)
- IIR_NatureDeclaration and IIR_SubnatureDeclaration (found in chapter on declarations)
- IIR_TerminalDeclaration (both in chapter on declarations)
- IIR_QuantityDeclaration with subclasses IIR_FreeQuantityDeclaration and IIR_BranchQuantityDeclaration (all found in chapter on declarations)

IIR_TerminalInterfaceDeclaration and IIRQuantityInterfaceDeclaration
(both found in chapter on declarations)

IIR_AcrossAttribute (found in chapter on names)

IIR_ThroughAttribute (found in chapter on names)

IIR_ReferenceAttribute (found in chapter on names)

IIR_ContributionAttribute (found in chapter on names)

IIR_DotAttribute (found in chapter on names)

IIR_IntegAttribute (found in chapter on names)

IIR_AboveAttribute (found in chapter on names)

IIR_BreakStatement (found in chapter on sequential statements)

IIR_SimultaneousStatement with the following subclasses:

- IIR_SimpleSimultaneousStatement (found in chapter on concurrent statements)
- IIR_ConcurrentBreakStatement (found in chapter on concurrent statements)
- IIR_SimultaneousIfStatement (found in chapter on concurrent statements)
- IIR_SimultaneousCaseStatement (found in chapter on concurrent statements)
- IIR_SimultaneousProceduralStatement (found in chapter on concurrent statements)
- IIR_SimultaneousNullStatement (found in chapter on concurrent statements)

19.7 Version 3.0 to Version 3.1

WARNING: An early version of 3.1 was distributed. The final version is dated 9/11/96.

A new IIR_TextLiteral class was created under IIR_Literal. IIR_Identifier, IIR_CharacterLiteral, IIR_StringLiteral and IIR_BitStringLiteral then became classes underneath IIR_TextLiteral. This changes provides unification for the classes consisting of variable-length character arrays (except for comments). The change is due to Dale Martin.

The means of getting an objects textual value was unified to the methods get_text() and get_text_length(). This change was request by several people.

An IIR_SimpleName class was added to IIR_Name in order to maintain consistency with the VHDL LRM. Use of this class is entirely optional since the name can always be replaced by its referent. This class was added at the request of the Savant project.

Name methods were added to IIR_DesignFile to denote the file name (change due to Robert Newshutz).

The IR_Text associated with a declaration is now referenced as a declarator, not an identifier. The identifier method was confusing in many context (an identifier for what). This change was reqested by several people.

The node IIR_GroupConstituent is no longer used and was removed (change due to Dale Martin).

The design unit being instantiated by an IIR_ComponentInstantiationStatement is now referenced by a parameter of type IIR rather than IIR_Name (change due to Malolan Chetlur and Dale Martin).

The IIR_AttributeSpecification changed in two ways. First get and set methods were added referencing the entity_class (as an identifier). The IIR_DesignatorList was renamed to an entity_name_list in order to maintain compatibility with the VHDL LRM (changes due to Tim McBrayer).

The IIR_PhysicalUnit unit name identifies an optional physical unit from which the multiplier is measured. Primary units have a unit_name pointing to this physical unit (This change was suggested by Tim McBrayer). The multiplier definition was modified and methods were added for the unit name as follows:

```
void
    set_unit_name(IIR_PhysicalUnit*      unit_name);
IIR_PhysicalUnit*
    get_physical_unit();
```

The public data elements in IIR_ComponentDeclaration were changed to have the more descriptive and consistent types IIR_GenericList and IIR_PortList rather than the more general IIR_InterfaceList (change due to Tim McBrayer).

The public data element representing the sensitivity list within a sensitized process had an obsolete type; the type was changed to IIR_DesignatorList (change due to Dale Martin).

19.8 Version 3.1 to 3.2

Subtype declarations associate a declarator with a (subtype) definition. Note that there are many subtype definitions which are anonymous, such as when a subtype constraint is applied to a type mark during the instantiation of an object. Each subtype declaration has includes a subtype indication, which AIRE represents as a (pointer to) a subtype indication. Generally the subtype indication's textual representation can be functionally reconstructed, however is there a need to retain the actual text string?

19.9 Version 3.2 to Version 4.0

IIR_Statement class was added as an intermediate class derived from IIR. The new IIR_Statement class becomes a parent class for IIR_SequentialStatement ,IIR_ConcurrentStatement, and IIR_SimultaneousStatement. The new non-terminal class includes reference to an IIR_Label declaration.

The class IIR_StatementList was added to represent statements containing sequential, concurrent or simultaneous statements.

IIR_PURE_PROCEDURAL and IIR_IMPURE_PROCEDURAL were added to IIR_PURITY type to accomodate VHDL-AMS.

IIR_SimpleSimultatenousStatement required two new methods in order to record the across and through tolerance.

Predefiend attributes IIR_ZOHAttribute, IIR_LTFAttribute and IIR_ZTFAttribute were added, corresponding to the addition of Laplace, frequency and Z-Domain transfer functions to VHDL-AMS.

The file_open_kind_expression belonging to an IIR_FileDeclaration became an IIR* (expression), generalizing the mode found in VHDL-87.

19.10 Version 4.0 to 4.1

For and While loop statements introduce a new declarative region. This requires a DeclarationList element declared as loop_declarations.

In IR_KIND enumeration, IIR_ACCESS_SUBTYPE_DEFINITION was missing, IIR_FREE_QUANTITY_DECLARATION and IIR_BRANCH_QUANTITY_DECLARATION was missing and IIR_QUANTITY_DECLARATION was in their place. IIR_COMPONENT_SPECIFICATION should be IIR_CONFIGURATION_SPECIFICATION and IIR_SENSITIZED_PROCESS_STATEMENT is missing. IIR_ENTITY_NAME_BY_NAME,IIR_ENTITY_NAME_BY_OTHERS,IIR_ENTITY_NAME_BY_ALL, were in the IR_KIND list but have not corresponing classes in name.doc

An intermediate class, IIR_NatureDefinition, was added as a peer to IIR_TypeDefinition and both scalar and composite natures were descended from it. References to nature and subnature which were incorrectly referred to as a type definition were corrected. The change was suggested by Chuck Swart and Rob Newshutz.

IIR_TerminalDeclaration, IIR_QuantityDeclaration, IIR_TerminalInterfaceDeclaration and IIR_QuantityInterfaceDeclaration were corrected.

A typo was corrected for IIR_PhysicalUnit so that the get and set methods matched.

The dispatch expression in an IIR_SimultaneousCaseStatement was changed to correctly refer to an expression and not condition.

The list of statements in an IIR_ArchitectureDeclaration was broadened to include both concurrent and simultaneous statements.

19.11 Version 4.1 to 4.2

Explicit subtypes (IIR_RecordSubtypeDefinition) were introduced of IIR_RecordTypeDefinition to accomodate resolution functions associated with subtype indications.

Explit subnatures were introduced of array and record natures in order to represent across and through tolerances.

IIR_NatureElementDeclaration and IIR_NatureElementDeclarationList was broken out from IIR_ElemenetDeclaration and IIR_ElementDeclarationList so that IIR_NatureRecordDefinition and IIR_NatureSubrecordDefinition would have elements of nature rather than type.

In the name chapter, representation was made for the new ‘contribution attribute (VHDL-AMS).

A new IR_SourceLanguage enumeration was introduced and associated with each design file. Rob noted that a single FIR couple represent information from more than one source language, and thus the magic number could no longer designate the language/

Numerous hypertext linkages were corrected.

References to IR_Kind enumerations were revised to refer to IR_..., rather than IIR_ so as to harmonize with the file intermediate representation.

19.12 Verion 4.5

The methods set_quantity_selector and get_quantity_selector were added to the IIR_BreakElement in order to track additions to the VHDL-AMS LRM. The type of both parameters was set to IIR* rather than IIR_Name* since the parameters may be IIR_QuantityDeclarations as well as forms of name.

A new class, IIR_AssociationElementByOthers, was added to cleanly support aggregates.

The elaboration chapter was generally circulated for the first time, including changes which substantially re-use pre-elaboration block structure. IR_Driver and IR_EffectiveValue were preserved from the previous version.

Reference was made to the EIA web site.

19.13 Version 4.6

Various textual corrections noted by Mitchell Perilstein were added (see email of 10 September). Still need to add section on naming convention near front and scan for additional errant spaces.

Within the IIR_Declaration hierarchy, IIR_TypeObjectDeclaration, IIR_NaturedObjectDeclaration, IIR_TypedInterfaceDeclaration and IIR_NaturedInterfaceDeclaration were added (Gordon Vreugdenhil at Analogy).

Reference to IIR_SignalNameList in IIR_WaitStatement was repalced with IIR_DesignatorList (Dr. Zainalabedin Navabi at Northeastern University).

Child classes were noted for IIR_Expression (Dr. Zainalabedin Navabi at Northeastern University).

S

T

V