

IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

IEEE Computer Society

Developed by the
Design Automation Standards Committee

IEEE Std 1685™-2022
(Revision of IEEE Std 1685-2014)

IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

Developed by the
Design Automation Standards Committee
of the
IEEE Computer Society

Approved 21 September 2022
IEEE SA Standards Board

Abstract: Conformance checks for eXtensible Markup Language (XML) data designed to describe electronic systems are formulated by this standard. The meta-data forms that are standardized include components, systems, bus interfaces and connections, abstractions of those buses, and details of the components including address maps, register and field descriptions, and file set descriptions for use in automating design, verification, documentation, and use flows for electronic systems. A set of XML schemas of the form described by the World Wide Web Consortium (W3C®) and a set of semantic consistency rules (SCRs) are included. A generator interface that is portable across tool environments is provided. The specified combination of methodology-independent meta-data and the tool-independent mechanism for accessing that data provides for portability of design data, design methodologies, and environment implementations.

Keywords: abstraction definitions, address space specification, bus definitions, design environment, EDA, electronic design automation, electronic system level, ESL, IEEE 1685™, implementation constraints, IP-XACT, register transfer level, RTL, SCRs, semantic consistency rules, TGI, tight generator interface, tool and data interoperability, use models, XML design meta-data, XML schema

This publication is dedicated to past editor and friend Joe Daniels, who passed away about 8 months after the team started to work on this revision in Accellera. Your work will live on as will your memory.

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2023 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 28 February 2023. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

“Accellera IP-XACT WG Comments on IEEE Std 1685-2014.” Material used with permission from Accellera Systems Initiative, © 2021.

AMBA is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

SystemC is a registered trademark of Accellera Systems Initiative.

Verilog is an abandoned trademark.

W3C is a registered trademark of the MIT.

XMLSpy is a registered trademark of Altova GmbH.

s

PDF: ISBN 978-1-5044-9448-9 STDGT25971
Print: ISBN 978-1-5044-9449-6 STDPD25971

IEEE Prohibits discrimination, harassment, and bullying.

*For more information, visit <https://www.ieee.org/about/corporate/governance/p9-26.html>
No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (<https://standards.ieee.org/ipr/disclaimers.html>), appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents are developed within IEEE Societies and subcommittees of IEEE Standards Association (IEEE SA) Board of Governors. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers are not necessarily members of IEEE or IEEE SA and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning this standard, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity, nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE is the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that the presenter's views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group. Statements made by volunteers may not represent the formal position of their employer(s) or affiliation(s).

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, consulting information, or advice pertaining to IEEE Standards documents.**

Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its Societies and subcommittees of the IEEE SA Board of Governors are not able to provide an instant response to comments, or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in evaluating comments or in revisions to an IEEE standard is welcome to join the relevant IEEE working group. You can indicate interest in a working group using the Interests tab in the Manage Profile & Interests area of the [IEEE SA myProject system](#).¹

Comments on standards should be submitted using the [Contact Us](#) form.²

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not constitute compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

¹ Available at: <https://development.standards.ieee.org/myproject-web/public/view.html#landing>.

² Available at: <https://standards.ieee.org/content/ieee-standards/en/about/contact/index.html>.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under US and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, neither IEEE nor its licensors waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate licensing fees, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400; <https://www.copyright.com/>. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every 10 years. When a document is more than 10 years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit [IEEE Xplore](#) or [contact IEEE](#).³ For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

Errata

Errata, if any, for all IEEE standards can be accessed on the [IEEE SA Website](#).⁴ Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in [IEEE Xplore](#). Users are encouraged to periodically check for errata.

³ Available at: <https://ieeexplore.ieee.org/browse/standards/collection/ieee>.

⁴ Available at: <https://standards.ieee.org/standard/index.html>.

Patents

IEEE Standards are developed in compliance with the [IEEE SA Patent Policy](#).⁵

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at <https://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

IMPORTANT NOTICE

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. IEEE Standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

⁵ Available at: <https://standards.ieee.org/about/sasb/patcom/materials.html>.

Participants

At the time the IEEE 1685 Standards Working Group (WG) completed this standard, it had the following entity-based membership:

Erwin de Kock, Chair
Jean-Michel Fernandez, Vice Chair
Richard Weber, Secretary
John Blyler, Technical Editor

<i>Organization Represented</i>	<i>Name of Representative</i>
Accellera Systems Initiative, Inc.	John Blyler
Advanced Micro Devices (AMD)	David Courtright
Arm Limited	Edwin Dankert
Arteris IP	Jean-Michel Fernandez
	Vincent Thibaut
	Grégoire Avot
	Benoit Lafage
Cadence Design Systems, Inc.	Eyal Herzberg
Cloudwalk Technology	Jun Li
Infineon Technologies AG	Michael Velten
Intel	Kamlesh Pathak
NVIDIA Corporation	Scott Venier
NXP Semiconductors	Erwin de Kock
Qualcomm, Inc.	David Cheng
Semifore, Inc.	Richard Weber
STMicroelectronics	Thomas Burg
Synopsys, Inc.	Mark Noll

The following members of the entity balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

1stCycle Corporation	Cadence Design Systems, Inc.	Qualcomm, Inc.
Accellera Systems Initiative, Inc.	Cloudwalk Technology	Semifore, Inc.
Advanced Micro Devices (AMD)	Infineon Technologies AG	Siemens Corporation
ARM, Ltd.	Intel	STMicroelectronics
Arteris IP	NVIDIA Corporation	Synopsys, Inc.
	NXP Semiconductors	

When the IEEE SA Standards Board approved this standard on 21 September 2022, it had the following membership:

David J. Law, *Chair*
Ted Burse, *Vice Chair*
Gary Hoffman, *Past Chair*
Konstantinos Karachalios, *Secretary*

Edward A. Addy
Ramy Ahmed Fathy
J. Travis Griffith
Guido R. Hiertz
Yousef Kimiagar
Joseph L. Koepfinger*
Thomas Koshy
John D. Kulick

Johnny Daozhuang Lin
Kevin Lu
Daleep C. Mohla
Andrew Myles
Damir Novosel
Annette D. Reilly
Robby Robson
Jon Walter Rosdahl

Mark Siira
Dorothy V. Stanley
Lei Wang
F. Keith Waters
Karl Weber
Sha Wei
Philip B. Winston
Daidi Zhong

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1685-2022, IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows.

The purpose of this standard is to provide the electronic design automation (EDA), semiconductor, electronic design intellectual property (IP) provider, and system design communities with a well-defined and unified specification for the meta-data that represents the components and designs within an electronic system. The goals of this specification are to enable delivery of compatible IP descriptions from multiple IP vendors; to improve the importing and exporting of complex IP bundles to, from, and between EDA tools for system on chip (SoC) design environments (DEs); to improve the expression of configurable IP by using IP meta-data; and to improve the provision of EDA vendor-neutral IP creation and configuration scripts (*generators*). The data and data access specification is designed to coexist and enhance the hardware description languages (HDLs) presently used by designers while providing capabilities lacking in those languages.

The SPIRIT Consortium, which originally developed the IP-XACT standard before merging with Accellera, was a consortium of electronic system, IP provider, semiconductor, and EDA companies. IP-XACT enables a productivity boost in design, transfer, validation, documentation, and use of electronic IP and covers components, designs, interfaces, and details thereof. The data specified by IP-XACT is extensible in locations specified in the schema.

IP-XACT enables the use of a unified structure for the meta specification of a design, components, interfaces, documentation, and interconnection of components. This structure can be used as the basis of both manual and automatic methodologies. IP-XACT specifies the tight generator interface (TGI) for access to the data in a vendor-independent manner.

This standardization project provides electronic design engineers with a well-defined standard that meets their requirements in structured design and validation and enables a step function increase in their productivity. This standardization project will also provide the EDA industry with a standard to which they can adhere and that they can support in order to deliver their solutions in this area.

Accellera has prepared a set of bus and abstraction definitions for several common buses. It is expected, over time, that standards groups and manufacturers who define buses will include IP-XACT eXtensible Markup Language (XML) bus and abstraction definitions in their set of deliverables. Until that time, and to cover existing useful buses, a set of bus and abstraction definitions for common buses has been created.

A set of reference bus and abstraction definitions allows many vendors who define IP using these buses to easily interconnect IP together. Accellera posts these definitions for use by its members, with no warranty of suitability, but in the hope that they will be useful. Accellera will, from time to time, update these files and, if a standards body wishes to take over the work of definition, will transfer that work to that body.

These reference bus and abstraction definition templates (with comments and examples) are available from the public area of the Accellera website.⁶

NOTE—Accellera IP-XACT WG Comments on IEEE Std 1685-2014 are acknowledged.

⁶Available at <http://www.accellera.org>.

Contents

1.	Overview.....	19
1.1	Scope.....	19
1.2	Purpose.....	19
1.3	Word usage.....	20
1.4	Design environment	20
1.5	IP-XACT–enabled implementations.....	25
1.6	Conventions used	26
1.7	Use of color in this standard.....	31
1.8	Contents of this standard.....	31
2.	Normative references.....	33
3.	Definitions, acronyms, and abbreviations.....	34
3.1	Definitions.....	34
3.2	Acronyms and abbreviations.....	39
4.	Interoperability use model	40
4.1	Roles and responsibilities.....	40
4.2	IP-XACT IP exchange flows.....	41
5.	Interface definition descriptions	43
5.1	Definition descriptions	43
5.2	Bus definition	43
5.3	Abstraction definition.....	45
5.4	Ports.....	46
5.5	Wire ports.....	47
5.6	Qualifiers.....	49
5.7	Wire port group	51
5.8	Wire port mode (and mirrored mode) constraints.....	52
5.9	Transactional ports	53
5.10	Transactional port group	54
5.11	Packets.....	55
5.12	Extending bus and abstraction definitions	56
5.13	Clock and reset handling.....	60
6.	Component descriptions	61
6.1	Component	61
6.2	Type definitions.....	63
6.3	Power domains	65
6.4	Interfaces	66
6.5	Interface interconnections	67
6.6	Complex interface interconnections.....	68
6.7	Bus interfaces	71
6.8	Indirect interfaces.....	79
6.9	Component channels	81
6.10	Modes.....	82

6.11	Address spaces	83
6.12	Memory maps.....	87
6.13	Remapping	99
6.14	Registers	100
6.15	Models.....	115
6.16	Component generators.....	151
6.17	File sets.....	153
6.18	Clear box elements	159
6.19	Clear box element reference.....	161
6.20	CPUs.....	162
6.21	Reset types.....	167
7.	Design descriptions.....	168
7.1	Design.....	168
7.2	Design component instances	170
7.3	Design interconnections	171
7.4	Active, hierarchical, monitored, and monitor interfaces.....	173
7.5	Design ad hoc connections	176
7.6	Port references.....	178
8.	Abstractor descriptions	180
8.1	Abstractor	180
8.2	Abstractor interfaces	182
8.3	Abstractor models	183
8.4	Abstractor views.....	184
8.5	Abstractor ports	185
8.6	Abstractor wire ports.....	186
8.7	Abstractor transactional port	189
8.8	Abstractor structured ports	190
8.9	Abstractor generators	193
9.	Type definitions descriptions.....	194
9.1	Type definitions.....	194
9.2	Field access policy definition	197
9.3	Enumeration definition.....	198
9.4	Field definition	199
9.5	Register definition	200
9.6	Register file definition.....	201
9.7	Address block definition	202
9.8	Bank definition.....	203
9.9	Memory map definition.....	205
9.10	Memory remap definition.....	207
10.	Generator chain descriptions	208
10.1	generatorChain	208
10.2	generatorChainSelector	210
10.3	generatorChain component selector	211
10.4	generatorChain generator	212

11.	Design configuration descriptions	214
11.1	Design configuration	214
11.2	designConfiguration	215
11.3	interconnectionConfiguration	216
11.4	abstractorInstance	217
11.5	viewConfiguration	218
12.	Catalog descriptions	219
12.1	catalog	219
12.2	ipxactFile	221
13.	Addressing	222
13.1	Calculating the bit address of a bit in a memory map or address space	222
13.2	Calculating the bus address at the bus interface	224
13.3	Calculating the address at the indirect interface	224
13.4	Address modifications of a channel	225
13.5	Address translation in a bridge	225
14.	Data visibility	226
14.1	Mapped address bits mask	226
14.2	Address modifications of an interconnection	226
14.3	Bit steering in a channel	226
14.4	Visibility of bits	227
	Annex A Bibliography	230
	Annex B Semantic consistency rules	231
B.1	Semantic consistency rule definitions	231
B.1.1	Compatibility of busDefinitions	231
B.1.2	Interface mode of a bus interface	231
B.1.3	Compatibility of abstractionDefinitions	231
B.1.4	Element referenced by configurableElementValue element	231
B.1.5	Memory mapping	231
B.1.6	Port connection equivalence class	232
B.1.7	Logical and physical ports	232
B.1.8	Addressable bus interface	232
B.1.9	Field connection graph	232
B.2	Rule listings	234
B.2.1	Cross-references and VLNVs	234
B.2.2	Interconnections	238
B.2.3	Channels, bridges, and abstractors	240
B.2.4	Monitor interfaces and monitor interconnections	243
B.2.5	Configurable elements	244
B.2.6	Ports	247
B.2.7	Registers	255
B.2.8	Memory maps	259
B.2.9	Addressing	260
B.2.10	Hierarchy	261
B.2.11	Hierarchy and memory maps	262

B.2.12	Constraints	262
B.2.13	Design configurations	264
B.2.14	Expressions	265
B.2.15	Access handles	268
Annex C	Common elements and concepts.....	269
C.1	accessHandles	269
C.1.1	simpleAccessHandle	269
C.1.2	slicedAccessHandle.....	270
C.1.3	portAccessHandle.....	271
C.1.4	sliceType	272
C.1.5	portSliceType	272
C.2	accessPolicies.....	273
C.2.1	Schema	273
C.2.2	Description	273
C.3	Access resolution	274
C.4	arrays.....	275
C.4.1	Configurable arrays with arrayId	275
C.4.2	Configurable arrays without arrayId	276
C.4.3	Memory arrays with stride	276
C.4.4	Field arrays with bit stride.....	277
C.5	assertions.....	278
C.5.1	Schema	278
C.5.2	Description	278
C.6	attributes.....	279
C.6.1	Schema	279
C.6.2	Description	280
C.7	complexBaseExpression	281
C.7.1	complexTiedValueExpression	282
C.7.2	qualifiedExpression	282
C.7.3	realExpression	283
C.7.4	signedLongintExpression	284
C.7.5	stringExpression	285
C.7.6	unresolvedStringExpression.....	285
C.7.7	unresolvedUnsignedBitExpression	286
C.7.8	unresolvedUnsignedPositiveIntExpression	286
C.7.9	unsignedBitExpression.....	287
C.7.10	unsignedBitVectorExpression	287
C.7.11	unsignedIntExpression	288
C.7.12	unsignedLongintExpression	289
C.7.13	unsignedPositiveIntExpression	290
C.7.14	unsignedPositiveLongintExpression	291
C.8	choices	292
C.8.1	Schema	292
C.9	configurableElementValues.....	293
C.9.1	Schema	293
C.9.2	Description	293
C.10	configurableLibraryRefType	294
C.10.1	Schema	294
C.10.2	Description	294
C.11	documentName group.....	295
C.11.1	Schema	295
C.11.2	Description	295

C.12	Endianness	296
C.13	fieldReferenceGroup.....	297
	C.13.1 Schema	297
	C.13.2 Description	297
C.14	fieldSliceReferenceGroup.....	299
	C.14.1 Schema	299
	C.14.2 Description	300
C.15	fileSetRef.....	300
	C.15.1 Schema	300
	C.15.2 Description	301
C.16	fileType.....	301
	C.16.1 Schema	301
	C.16.2 Description	301
C.17	indices	302
	C.17.1 Schema	302
	C.17.2 Description	302
C.18	libraryRefType.....	303
	C.18.1 Schema	303
	C.18.2 Description	303
C.19	Name groups	304
	C.19.1 nameGroup group.....	304
	C.19.2 nameGroupNMTOKEN group.....	305
	C.19.3 nameGroupOptional group.....	306
	C.19.4 nameGroupPort group	307
	C.19.5 nameGroupString group.....	308
C.20	nameValuePairType.....	308
	C.20.1 Schema	308
	C.20.2 Description	309
C.21	parameters.....	309
	C.21.1 Schema	309
	C.21.2 Description	309
C.22	partSelect	310
	C.22.1 Schema	310
	C.22.2 Description	310
C.23	pathSegments	311
	C.23.1 pathSegment.....	311
	C.23.2 portPathSegment	311
C.24	Power constraints.....	312
	C.24.1 transactionalPowerConstraints	312
	C.24.2 wirePowerConstraints	312
C.25	range.....	313
	C.25.1 Schema	313
	C.25.2 Description	313
C.26	Vectors	314
	C.26.1 vectors	314
	C.26.2 extendedVectorsType.....	314
C.27	vendorExtensions.....	315
	C.27.1 Schema	315
	C.27.2 Description	315
C.28	versionedIdentifier group.....	315
	C.28.1 Schema	315
	C.28.2 Description	316
	C.28.3 Sorting and comparing version elements	316
	C.28.4 Version control	317

C.29	viewRef.....	318
C.29.1	Schema	318
C.29.2	Description	318
C.30	xml:id	318
C.31	Component vs. Abstraction Definition Ports	318
Annex D	Types.....	320
D.1	boolean.....	320
D.2	float	320
D.3	ID or IDREF	320
D.4	instancePath	320
D.5	integer	320
D.6	libraryRefType.....	320
D.7	Name	320
D.8	NMOKEN	321
D.9	NMOKENS	321
D.10	portName	321
D.11	ipxactURI.....	321
D.12	string	321
D.13	token.....	321
Annex E	SystemVerilog expressions.....	322
E.1	Overview.....	322
E.2	Data-types	322
E.2.1	bit data type	322
E.2.2	byte data type	322
E.2.3	shortint data type	323
E.2.4	int data type	323
E.2.5	longint data type	323
E.2.6	shortreal data type	323
E.2.7	real data type	323
E.2.8	string data type	323
E.2.9	Signed and unsigned data types	324
E.2.10	Unresolved data types	324
E.3	Assignments.....	324
E.3.1	Single value assignment	324
E.3.2	Parameter type.....	324
E.3.3	Parameter signing	325
E.3.4	Vector assignment	325
E.3.5	Array assignment.....	325
E.3.6	Identifiers	326
E.3.7	Identifier references.....	326
E.4	Operators.....	328
E.5	Functions.....	329
E.5.1	Integer function	329
E.5.2	Real functions.....	329
E.5.3	String function.....	330
E.5.4	IP-XACT specific functions.....	332
E.5.5	IP-XACT specific escape sequences	334
E.6	Expression language formal syntax (BNF).....	335
E.6.1	Declarations: declaration data types	335
E.6.2	Behavioral statements: Case statements.....	335

E.6.3	Expressions.....	336
E.6.4	General: Identifiers.....	337
E.7	SystemVerilog conversion steps.....	338
E.7.1	Convert parameter	338
E.7.2	Convert expression	338
E.8	SystemVerilog reference.....	338
Annex F	Tight generator interface.....	341
F.1	Method of communication.....	341
F.2	Generator invocation.....	341
F.2.1	Resolving the URL.....	342
F.2.2	Example.....	342
F.3	TGI API	343
F.3.1	TGI fault codes.....	344
F.3.2	Administrative commands.....	344
F.3.3	Return values.....	345
F.4	IDs and configurable values	345
F.5	TGI messages.....	346
F.6	Vendor attributes.....	346
F.7	TGI calls	346
F.7.1	Category index	346
F.7.2	Abstraction definition (BASE).....	348
F.7.3	Abstraction definition (EXTENDED).....	354
F.7.4	Abstractor (BASE)	365
F.7.5	Abstractor (EXTENDED)	366
F.7.6	Access Policy (BASE)	370
F.7.7	Access handle (BASE)	370
F.7.8	Access handle (EXTENDED)	371
F.7.9	Access policy (BASE).....	372
F.7.10	Access policy (EXTENDED).....	379
F.7.11	Address space (BASE)	386
F.7.12	Address space (EXTENDED)	389
F.7.13	Array (BASE).....	392
F.7.14	Array (EXTENDED).....	393
F.7.15	Assertion (BASE).....	395
F.7.16	Assertion (EXTENDED)	395
F.7.17	Bus definition (BASE)	396
F.7.18	Bus definition (EXTENDED)	397
F.7.19	Bus interface (BASE).....	399
F.7.20	Bus interface (EXTENDED).....	406
F.7.21	CPU (BASE)	413
F.7.22	CPU (EXTENDED)	415
F.7.23	Catalog (BASE).....	417
F.7.24	Catalog (EXTENDED)	418
F.7.25	Choice (BASE).....	421
F.7.26	Choice (EXTENDED)	422
F.7.27	Clearbox (BASE)	422
F.7.28	Clearbox (EXTENDED)	423
F.7.29	Component (BASE)	424
F.7.30	Component (EXTENDED)	427
F.7.31	Configurable element (BASE)	436
F.7.32	Configurable element (EXTENDED)	437
F.7.33	Constraint (BASE)	438

F.7.34	Constraint (EXTENDED)	441
F.7.35	Constraint Set (BASE)	443
F.7.36	Constraint Set (EXTENDED)	444
F.7.37	Design (BASE).....	444
F.7.38	Design (EXTENDED).....	448
F.7.39	Design configuration (BASE)	454
F.7.40	Design configuration (EXTENDED)	458
F.7.41	Driver (BASE).....	461
F.7.42	Driver (EXTENDED).....	467
F.7.43	Element attribute (BASE)	470
F.7.44	Element attribute (EXTENDED)	485
F.7.45	File builder (BASE)	508
F.7.46	File builder (EXTENDED)	514
F.7.47	File set (BASE)	520
F.7.48	File set (EXTENDED)	525
F.7.49	Generator (BASE)	531
F.7.50	Generator (EXTENDED).....	533
F.7.51	Generator chain (BASE)	535
F.7.52	Generator chain (EXTENDED)	537
F.7.53	Indirect interface (BASE).....	539
F.7.54	Indirect interface (EXTENDED).....	545
F.7.55	Instantiation (BASE)	551
F.7.56	Instantiation (EXTENDED).....	555
F.7.57	Memory map (BASE)	559
F.7.58	Memory map (EXTENDED)	569
F.7.59	Miscellaneous (BASE).....	580
F.7.60	Miscellaneous (EXTENDED).....	583
F.7.61	Module parameter (BASE).....	585
F.7.62	Module parameter (EXTENDED).....	586
F.7.63	Name group (BASE)	586
F.7.64	Name group (EXTENDED)	587
F.7.65	Parameter (BASE).....	588
F.7.66	Parameter (EXTENDED).....	589
F.7.67	Port (BASE)	590
F.7.68	Port (EXTENDED)	604
F.7.69	Port map (BASE)	626
F.7.70	Port map (EXTENDED)	628
F.7.71	Power (BASE).....	630
F.7.72	Power (EXTENDED).....	631
F.7.73	Register (BASE).....	633
F.7.74	Register (EXTENDED).....	644
F.7.75	Register file (BASE)	654
F.7.76	Register file (EXTENDED)	656
F.7.77	Slice (BASE)	658
F.7.78	Slice (EXTENDED)	663
F.7.79	Top element (BASE)	667
F.7.80	Top element (EXTENDED)	668
F.7.81	Type definitions (BASE).....	670
F.7.82	Type definitions (EXTENDED).....	677
F.7.83	Vector (BASE)	686
F.7.84	Vector (EXTENDED)	686
F.7.85	Vendor extensions (BASE)	687
F.7.86	Vendor extensions (EXTENDED)	687
F.7.87	View (BASE)	687

F.7.88	View (EXTENDED)	689
F.7.89	All ID types	690
Annex G	External bus with an internal/digital interface	693
G.1	Example: Ethernet interfaces	693
G.2	Example: I2C bus.....	694
Annex H	Bridges and channels	695
H.1	Transparent bridge	696
H.2	Opaque bridge.....	697
H.2.1	Without an address space segment reference	697
H.2.2	With an address space segment reference	698
H.2.3	Effect of an initiator interface address space base address	699
H.3	Channel with address remapping.....	701
H.4	Channel with bit steering	702
Annex I	Examples.....	705
I.1	abstractionDefinition - RTL.....	705
I.2	abstractionDefinition - TLM.....	707
I.3	abstractor.....	708
I.4	busDefinition	709
I.5	catalog	710
I.6	component.....	712
I.7	design	736
I.8	designConfiguration.....	739
I.9	fieldAccessPolicyDefinitions.....	740
I.10	generatorChain.....	743
I.11	typeDefinitions.....	744

IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

1. Overview

This clause explains the scope and purpose of this standard; gives an overview of the basic concepts, major semantic components, and conventions used in this standard; and summarizes its contents.

1.1 Scope

This standard describes an eXtensible Markup Language (XML) schema⁷ for meta-data documenting *intellectual property* (IP) used in the development, implementation, and verification of electronic systems. This schema provides both a standard method to document IP that is compatible with automated integration techniques and a standard method (generators) for linking tools into a *system development* framework, enabling a more flexible, optimized development environment. Tools compliant with this standard will be able to interpret, configure, integrate, and manipulate IP blocks that comply with the IP meta-data description. The standard is independent of any specific design processes. It does not cover behavioral characteristics of the IP that are not relevant to integration.

1.2 Purpose

This standard enables the creation and exchange of IP in a highly automated design environment.

⁷Information on references can be found in [Clause 2](#).

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (shall equals is required to).^{8, 9}

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (should equals is recommended that).

The word *may* is used to indicate a course of action permissible within the limits of the standard (may equals is permitted to).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (can equals is able to).

1.4 Design environment

The IP-XACT specification is a mechanism to express and exchange information about design IP and its required configuration.¹⁰ While the IP-XACT description formats are the core of this standard, describing the IP-XACT specification in the context of its basic use model, the design environment (DE), more readily depicts the extent and limitations of the semantic intent of the data. The DE coordinates a set of tools and IP, or expressions of that IP (e.g., models), through the creation and maintenance of meta-data descriptions of the system on chip (SoC) so that its system design and implementation flows are efficiently enabled and reuse centric.

The use of the IP-XACT-specified formats and interfaces are shown in **bold** in [Figure 1](#) and described in the following subclauses.

⁸The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.

⁹The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

¹⁰IP-XACT uses the World Wide Web Consortium (W3C®) standard for the XML version 1.0 data (<http://www.w3.org/TR/REC-xml/>). The valid format of that XML data is described in a *schema* by using the Schema Description Language described therein. W3C is a registered trademark of the World Wide Web Consortium. In addition to XML, many configurability aspects of IP-XACT are inspired from the IEEE 1800 SystemVerilog standard.

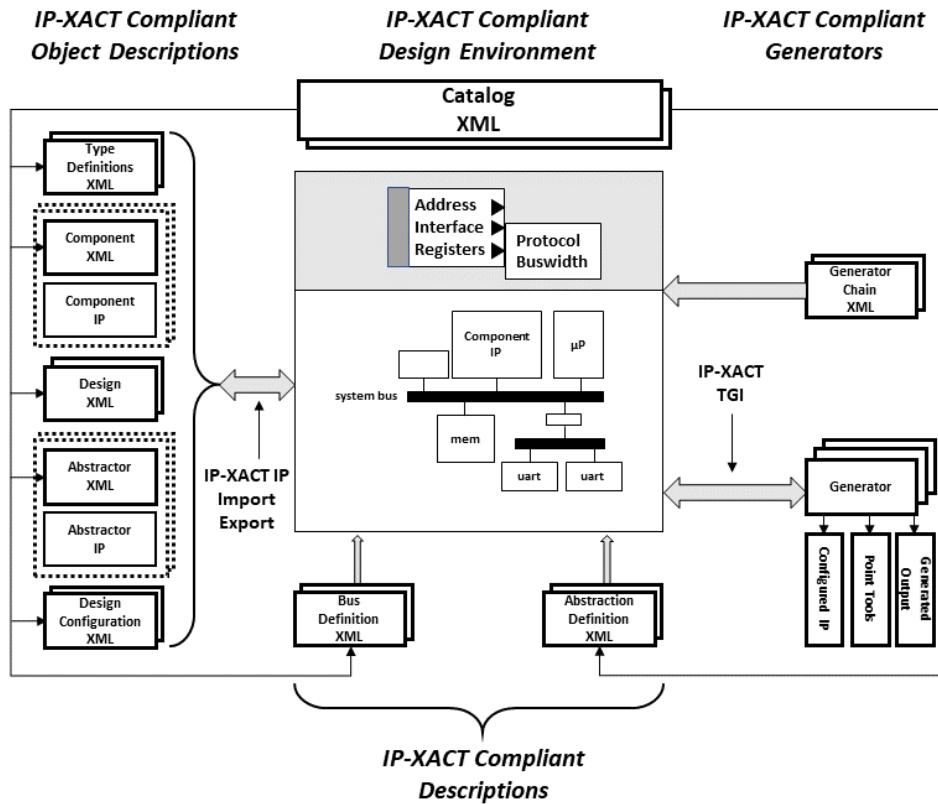


Figure 1—IP-XACT design environment

1.4.1 IP-XACT design environment

A DE enables the designer to work with IP-XACT design IP through a coordinated front-end and IP design database. These tools create and manage the top-level meta-description of system design and may provide two basic types of services: *design capture*, which is the expression of design configuration by the IP provider and design intent by the IP user, and *design build*, which is the creation of a design (or design model) to those intentions.

As part of design capture, a system design tool shall recognize the structure and configuration options of imported IP. In the case of *structure*, this implies both the structure of the design (e.g., how specific pin-outs refer to lines in the hardware description language (HDL) code) as well as the structure of the IP package (e.g., where design descriptions and related generators are provided in the packaged IP data-structure). In the case of *configuration*, this is the set of options for handling the imported IP (e.g., setting the base address and offset, bus width) that may be expressed as configurable parameters in the IP-XACT meta-data.

As part of design build, generators may be provided internally by a system design tool to achieve the required IP integration or configuration, or they may be provided externally (e.g., by an IP provider) and launched by the system design tool as appropriate.

The *system design tool set* defines a DE where the support for conceptual context and management of IP-XACT meta-data resides. However, the IP-XACT specifications make no requirements upon system design

tool architecture or a tool's internal data structures. To be considered IP-XACT enabled, a system design tool shall support the import/export of IP expressed with valid IP-XACT meta-data for both component IP and designs, and it needs to support the tight generator interface (TGI) for interfacing with external generators (to the DE).

1.4.2 IP-XACT object descriptions

The IP-XACT schema is the core of the IP-XACT specification. There are nine top-level schema definitions. Each schema definition can be used to create object descriptions of the corresponding types:

- A *bus definition* description defines the type attributes of a bus.
- An *abstraction definition* description defines the representation attributes of a bus.
- A *typeDefinitions* description defines memory map-related elements, including registers, of a component.
- A *component* description defines an IP or interconnect structure.
- A *design* description defines the configuration of and interconnection between components.
- An *abstractor* description defines an adaptor between interfaces of two different abstractions.
- A *generator chain* description defines the grouping and ordering of generators.
- A *design configuration* description defines additional configuration information for a generator chain or design description.
- A *catalog* description provides a mapping between IP-XACT VLNVs (see [1.4.3](#)) and the physical location of the IP-XACT file defining the IP-XACT object with the given VLVN.

1.4.3 Object interactions

An object description contains a unique identifier in the header. The identifier in IP-XACT terms is called a *VNV* after the four elements that define its value: vendor, library, name, and version. See [C.28](#) for further details on a VNV. This VNV is used to create a reference from one description to another. The links between these objects are illustrated in [Figure 2](#). The arrows (A → B) illustrate a reference of one object to another (e.g., reference of object B from object A).

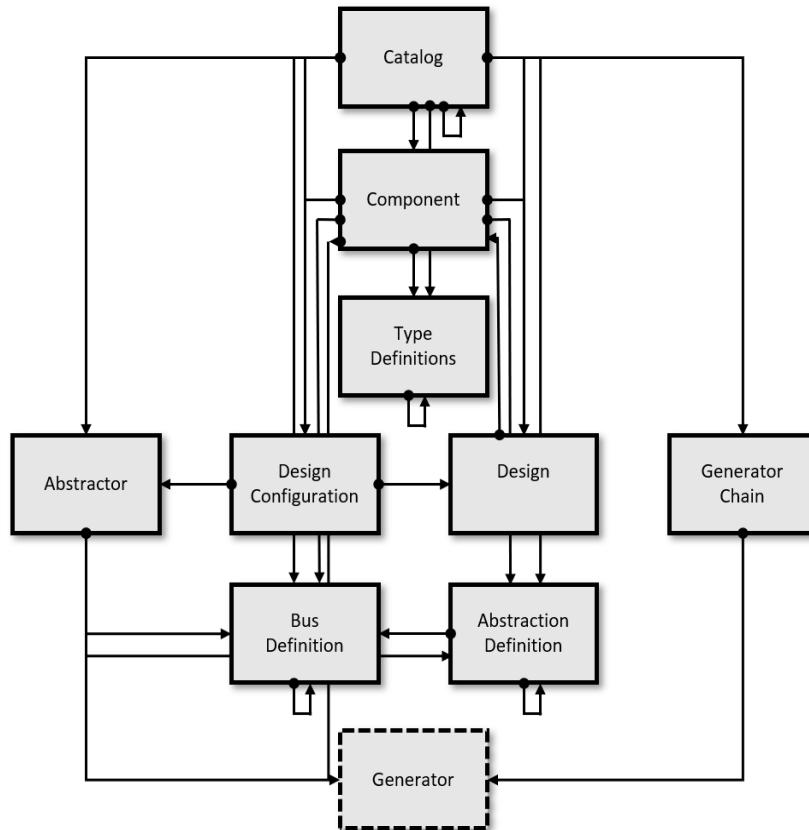


Figure 2—IP-XACT object interactions

1.4.4 IP-XACT generators

Generators are executable objects (e.g., scripts or binary programs) that may be integrated within a DE (referred to as *internal*) or provided separately as an executable (referred to as *external*). Generators may be provided as part of an IP package (e.g., for configurable IP, such as a bus-matrix generator) or as a way of wrapping point tools for interaction with a DE (e.g., an external design netlister, external design checker); see [1.5.2](#).

An internal generator may perform a wide variety of tasks and may access IP-XACT-compliant meta-data by any method a DE supports. IP-XACT does not describe these protocols.

An *external generator* (often referred to as a *TGI generator*) is an executable program or script invoked from within a DE to query or configure design descriptions and their related component and abstractor descriptions. External generators can use the TGI to access their IP-XACT meta-data descriptions (as currently loaded into the DE) and perform the various operations associated with those descriptions. In addition, external generators shall only *operate* upon IP-XACT-compliant meta-data through the defined TGI; see [1.4.6](#).

Generators can be referenced from a component, abstractor, or generator chain description. Generators can also be grouped and ordered in generator chain descriptions and in chain descriptions contained inside other chain descriptions. This sequencing of generators is *critical* for providing script-based support for SoC flow creation.

1.4.5 IP-XACT design environment interfaces

There are two interfaces expressed in [Figure 1](#): from the DE to the external IP libraries and from the DE to the generators. In the former case, the IP-XACT specifications are *neutral* regarding the design tool interfaces to IP repositories. Being able to read and write IP with IP-XACT meta-data is required; however, the *formal interaction* between an external IP repository and a DE is not specified. In the latter case, the interface between the DE and a generator is the TGI; see [1.4.6](#).

1.4.6 Tight generator interface

The *tight generator interface* (TGI) is the method a generator uses to access a design or component description in a DE-independent and generator-language-independent manner. Therefore, a generator running on two different DEs produces the same results. The DE and the generator can communicate with each other by sending messages utilizing the Simple Object Access Protocol (SOAP)¹¹ specified in the Web Services Description Language (WSDL).¹² SOAP or REST¹³ provide a simple means for sending XML-format messages using the Hypertext Transfer Protocol (HTTP) or other transport protocols. IP-XACT supports using HTTP or a file protocol.

The messages passed between the generator and the DE allow the generator to get all information about the design interconnections (which contain components and abstractors), provide set information for any configurable elements in a component or abstractor, and make simple modifications of the design description. For additional details on the DE generator invocation and the messages passed between the generator and the DE, see [Annex F](#).

1.4.7 Design intellectual property

IP-XACT is structured around the concept of IP reuse. *Electronic design intellectual property*, or IP, is a term used in the electronic design automation (EDA) community to refer to a reusable collection of design specifications that represent the behavior, properties, and/or description of the design in various media. The name IP is partially derived from the common practice of considering a collection of this type to be the intellectual property of one party. Both hardware and software collections are encompassed by this term.

These collections may include the following:

- a) Design objects—This collection can include the following:
 - 1) Transaction-level modeling (TLM) descriptions: SystemC® and SystemVerilog¹⁴
 - 2) Fixed HDL descriptions: Verilog®, VHDL¹⁵
 - 3) Configurable HDL descriptions (e.g., bus-fabric generators)
 - 4) Design models for register transfer level (RTL) and transactional simulation (e.g., compiled core models)
 - 5) HDL-specified verification IP (VIP) (e.g., basic stimulus generators and checkers)
- b) IP views—This collection is a list of different views (levels of description and/or languages) to describe the IP object. In IP-XACT, these views include the following:
 - 1) Design view: RTL Verilog or VHDL, flat or hierarchical components
 - 2) Simulation view: model views, targets, simulation directives, etc.
 - 3) Documentation view: standard, user guide, etc.

¹¹Available from the W3C Web site at <http://www.w3.org/TR/soap12-part1/>.

¹²Available from the W3C Web site at <http://www.w3.org/TR/wsdl>.

¹³Available from the W3C Web site at <https://www.w3.org/TR/ws-arch/#relwwwrest>

¹⁴SystemC is a registered trademark of the Accellera Systems Initiative.

¹⁵Verilog is an abandoned trademark.

IP-XACT XML meta-data descriptions provide a standardized way of collecting much of the structural information contained in the file sets. IP-XACT also can contain the information that identifies the appropriate files included in a collection to be used for different parts of the design process.

1.5 IP-XACT-enabled implementations

Complying with the rules outlined in this subclause allows providers of tools, IP, or generators to class their products as *IP-XACT enabled*. Conversely, any violation of these rules removes that naming right. This subclause first introduces the set of metrics for measuring the valid use of the specifications. It then specifies when those validity checks are performed by the various classes of products and providers: DEs, point tools, IPs, and generators.

- a) Parse validity
 - 1) Parsing correctness: Ability to read all IP-XACT descriptions.
 - 2) Parsing completeness: Cannot require information that could be expressed in an IP-XACT format to be specified in a non-IP-XACT format. Processing of all information present in an IP-XACT document is not required.
- b) Description validity
 - 1) Schema correctness: IP is described using XML files that conform to the IP-XACT schema.
 - 2) Usage completeness: Extensions to the IP-XACT schema shall be used only to express information that cannot otherwise be described in IP-XACT.
- c) Semantic validity
 - 1) Semantic correctness: Adheres to the semantic interpretations of IP-XACT data described in this standard.
 - 2) Semantic completeness: Obeys all the semantic consistency rules (SCRs) described in [Annex B](#).

These validity rules can be combined with the product class-specific rules to cover the full IP-XACT-enabled space. The following subclauses describe the rules a provider has to check to claim a product is IP-XACT enabled.

An IP-XACT-enabled DE or point tool may read descriptions based on multiple versions of the IP-XACT schema. If the DE or point tool does provide this capability, the effect shall be as if all of the descriptions had been translated to the highest schema version supported by the given tool. This is required for semantic consistency. Schema version translation can be done in a number of different ways, but the most common is to leverage the eXtensible Stylesheet Language Transformations (XSLT)¹⁶ provided with the IP-XACT schema image. In addition, a DE or point tool may preserve information in the initial description for use outside of the scope of the IP-XACT specification.

1.5.1 Design environments

An IP-XACT-enabled DE shall

- Follow the parse validity requirements shown in [1.5](#).
- Create IP that is IP-XACT enabled.
- Modify any existing IP-XACT descriptions without losing any preexisting information. In particular, it shall preserve any vendor extension data included in the existing IP-XACT description.
- Be able to invoke IP-XACT-enabled generators with **apiType** of **none** (see [6.16.2](#)).

¹⁶Available from the W3C Web site at <http://www.w3.org/TR/xslt>.

An IP-XACT–TGI–enabled DE shall be considered as enabled as follows:

- For the base TGI, if it supports all generators utilizing the TGI_2022_BASE **apiType** (see [6.16.2](#)).
- For the extended TGI, if it supports all generators utilizing the TGI_2022_BASE or TGI_2022_EXTENDED **apiTypes** (see [6.16.2](#)).

1.5.2 Point tools

A point tool is a tool that has a particular, rather than a general, set of capabilities. In contrast to an IP-XACT–enabled DE (see [1.5.1](#)), an IP-XACT–enabled point tool shall

- Follow the parse validity requirements shown in [1.5](#).
- Create IP that is IP-XACT enabled.
- Modify any existing IP-XACT descriptions while trying not to lose any preexisting information. In particular, it shall preserve any vendor extension data included in the existing IP-XACT description.

1.5.3 IPs

An IP-XACT–enabled IP shall

- Have an IP-XACT description that follows the description and semantic validity requirements shown in [1.5](#).
- Use IP-XACT–enabled generators for any generators associated with this IP.

XML descriptions compliant with IP-XACT shall provide a namespace reference to the `index.xsd` schema file, not to any of the other files in the release.

1.5.4 Generators

An IP-XACT–enabled generator shall

- Create IP that is IP-XACT enabled.
- Modify any existing IP-XACT descriptions without losing any preexisting information. In particular, it shall preserve any vendor extension data included in the existing IP-XACT description.
- Communicate with the DE that invoked it only through the IP-XACT TGI (see [Annex F](#)).

1.6 Conventions used

The conventions used throughout the document are included here.

IP-XACT is case-sensitive.

1.6.1 Visual cues (meta-syntax)

Bold shows required keywords and/or special characters, e.g., **addressSpace**. For the initial definitional use (per element), keywords are shown in **boldface-red text**, e.g. **bitsInLau** (see also [1.7](#)).

Bold italics shows group names or data types, e.g., **nameGroup** or **boolean**. For definitions of types, see [Annex D](#).

Courier shows examples, external command names, directories and files, etc., e.g., address 0x0 is on D[31:0].

1.6.2 Notational conventions

The keywords *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this standard are to be interpreted as described in [1.3](#) and IETF RFC-2119 [[B5](#)].¹⁷ When a conflict between the two exists, the use in [1.3](#) takes precedence.

1.6.3 Syntax examples

Any syntax examples shown in this standard are for information only and are intended only to illustrate the use of such syntax (see also [Annex I](#)).

1.6.4 Graphics used to document the schema

The W3C Web site¹⁸ specifies the XML schema language used to define the IP-XACT XML schemas. Normative details for compliance to the IP-XACT standard are contained in the schema files. Within this document, pictorial representations of the information in the schema files *illustrate* the structure of the schema and *define* any constraints of the standard. With the exception of visibility issues, the information in the figures and the schema files is intended to be identical (for a fully annotated version of the schema files; see IP-XACT Schema [[B8](#)]). Where the figures and schema are in conflict, the XML schema file shall take precedence.¹⁹

1.6.4.1 Elements and attributes

The *element* is the fundamental building block on which this standard is based. An element may be either a *leaf element*, which is a container for information, or a *branch element*, which may contain further branch elements or leaf elements.

A leaf or branch element may also contain *attributes*. Attributes are containers for information within the containing element.

1.6.4.2 Types

A *type* is a designation of the format for the contents of an element or attribute. There are two different styles of types that can be defined. A type may be assigned to a leaf element or an attribute. This type is called a *simpleType* and defines the format of data that may be stored in this container. A type may also be assigned to a branch element. This type is called a *complexType* and defines further elements and attributes contained in the branch element.

1.6.4.3 Groups

A group is a collection of elements or attributes, which allow the same collection of items to be referenced consistently in many places. There are two different types of groups that can be defined. A *group* is a combination of leaf or branch elements; an *attributeGroup*, a simple list of attributes. The names assigned to either group have no representation in the resulting description.

¹⁷The number in brackets correspond to the numbers of the bibliography in [Annex A](#).

¹⁸Available from the W3C Web site at <https://www.w3.org/TR/REC-xml/>.

¹⁹The graphics for this document have been generated by taking “screen-shots” of the various files as they are displayed in Altova’s XML environment XMLSpy®. XMLSpy is a registered trademark of Altova GmbH. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of this product. Equivalent products may be used if they can be shown to lead to the same results.

1.6.4.4 Namespace

Each element, attribute, type, or group has a name, which is preceded by a namespace-prefix and separated from the name by a colon (:). For the examples in [1.6.4.5](#), xyz is used as the namespace-prefix for all of the items whereas this standard uses ipxact. Within the text of this standard, the namespace-prefix is not written when describing an item; it is shown only in examples.

1.6.4.5 Diagrams

The diagrams used throughout this standard graphically detail the organization of elements and attributes.

NOTE—For brevity, the `xml:id` attribute (see [C.30](#)) has been removed from all diagrams.²⁰

1.6.4.5.1 Elements and sequences

[Figure 3](#) shows the sequence-compositor. At the left is a branch element, **element1**, with some descriptive text below. **element1** is connected to a sequence-compositor. The sequence-compositor defines the order the subelements appear in the branch element. **subElement1** shall appear first inside of **element1**. This is followed by **subElement2**, **subElement3**, **subElement4**, and **subElement5** before closing **element1**.

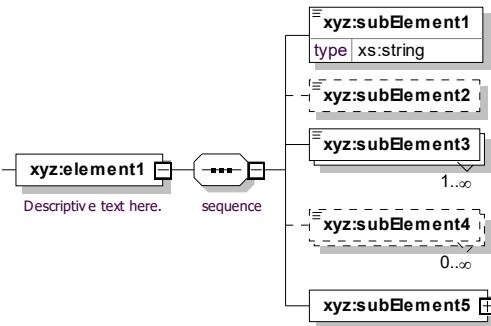


Figure 3—Sequence-compositor

- a) **subElement1** is a mandatory element, as indicated by the solid line of the containing box. The type of the data contained in this element is set to *string*, and it has a default value of ipxact if the element is present but left empty.
- b) **subElement2** is an optional element, as indicated by the dashed-line of the containing box.
- c) **subElement3** is an mandatory element that may appear multiple times, indicated by the double-solid line of the containing box. The number of times the element may appear is indicated by the range of the numbers listed below the element.
- d) **subElement4** is an optional element that may appear multiple times, as indicated by the double-dashed line of the containing box. The number of times the element may appear is indicated by the range of the numbers listed below the element.
- e) **subElement5** is an mandatory branch element that contains further elements inside, as indicated by the small plus sign (+) in the small box on the right.

²⁰Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

[Figure 4](#) shows variations of a sequence-compositor. **root1** is connected to an optional sequence-compositor, as indicated by the symbol being drawn with a dashed line. **element1** may appear first inside of **root1**; if it does, it shall be followed by **element2**. Each subelement is connected to a sequence-compositor.

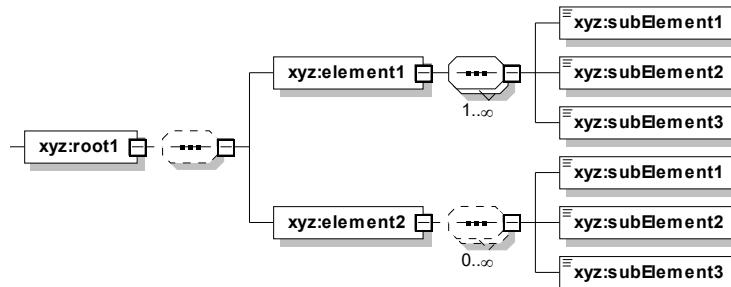


Figure 4—Sequence-compositor variations

- **element1** may contain one or more of the following sequences in the following order: **subElement1** and **subElement2** and **subElement3**. The number of times the sequence-compositor may appear is indicated by the range of the numbers listed below the symbol. If the range is greater than 1, the sequence-compositor symbol is drawn with double lines.
- **element2** is optional and may contain zero or more of the following sequences in the following order: **subElement1** and **subElement2** and **subElement3**. The number of times the sequence-compositor may appear is indicated by the range of the numbers listed below the symbol. If the range starts at 0 and the maximum is greater than 1, the sequence-compositor symbol is drawn with double-dashed lines.

1.6.4.5.2 Elements and choices

[Figure 5](#) shows the variations of the choice-compositor. **root** is connected to a choice-compositor. The choice-compositor specifies that one of the elements on the right side shall be chosen. **root** may contain one of the following: **element1**, **element2**, or **element3**. Each subelement is connected to a choice-compositor.

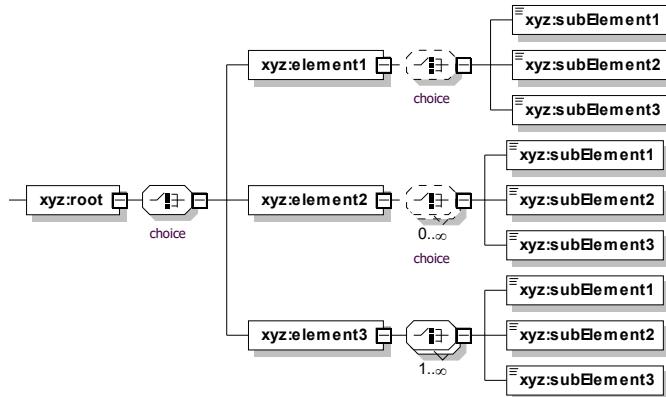


Figure 5—Choice-compositor variations

- element1** may contain one of the following: **subElement1**, **subElement2**, or **subElement3**, as indicated by the symbol being drawn with a dashed line.
- element2** may contain any (0 or more) of the following: **subElement1**, **subElement2**, or **subElement3** in any order. The number of times the choice-compositor may appear is indicated by

the range of the numbers listed below the symbol. If the range starts at 0, the choice-compositor is drawn with dashed lines.

- c) **element3** may contain one or more of the following: **subElement1**, **subElement2**, or **subElement3** in any order. The number of times the choice-compositor may appear is indicated by the range of the numbers listed below the symbol. If the range is greater than 1, the choice-compositor is drawn with double lines.

1.6.4.5.3 Elements, attributes, groups, and attributeGroups

Figure 6 shows the use of attributes, groups, and attributeGroups. **element1** contains two attributes, shown in the tab-shaped box labeled *attributes*. **attribute1** is optional, as indicated by the dashed containing box, and is of type **integer** with a default value of 7 if the attribute is not present. **attribute2** is a required attribute, as indicated by the solid containing box, and is of type **boolean** with no default. The ordering in which **attribute1** and **attribute2** appear inside **element1** is irrelevant.

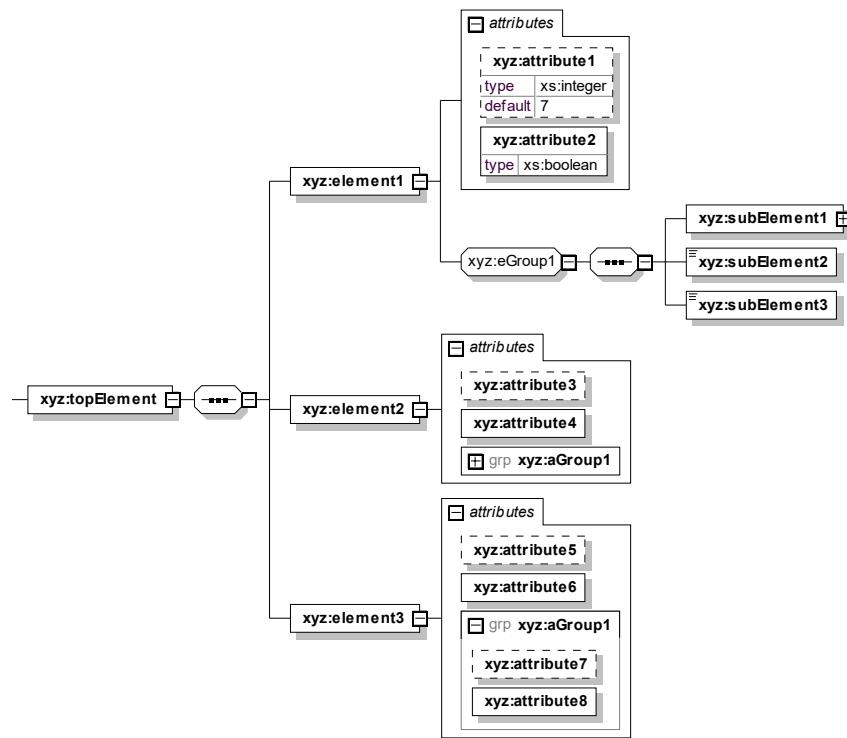


Figure 6—Attributes, groups, and attributeGroups

- a) **eGroup1** is an element group inside **element1**. This group contains three subelements, and the group symbol can be replaced by a solid line. The name of the group has no representation in the resulting output description. An element group can be optional, as indicated by a dashed outline (not shown), and it can also have a range, as indicated by numbers below the group symbol (not shown).
- b) **aGroup1** is an **attributeGroup** inside **element2** and **element3**. This **attributeGroup** contains two attributes, **attribute7** and **attribute8**. Inside **element2**, the **attributeGroup** is shown in its collapsed form, as indicated by the small plus sign (+) inside the small box. Inside **element3**, the **attributeGroup** is shown in its expanded form, as indicated by the small minus sign (-) inside the small box. **element2** contains four attributes: **attribute3**, **attribute4**, **attribute7**, and **attribute8**. **element3** also contains four attributes: **attribute5**, **attribute6**, **attribute7**, and **attribute8**. The name of the **attributeGroup** has no representation in the resulting description.

1.6.4.5.4 Wildcards

Figure 7 shows the use of wildcards. A *wildcard* is depicted by the rounded box with the **any ##any** text. Wildcards indicate that any well-formed attribute or element may be inserted into the containing element.

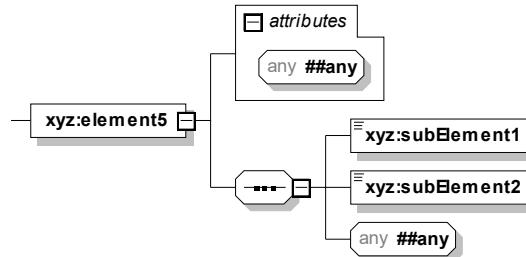


Figure 7—Wildcards

1.7 Use of color in this standard

This standard uses a minimal amount of color to enhance readability. The coloring is not essential and does not affect the accuracy of this standard when viewed in pure black and white. The places where color is used are the following:

- Cross references that are hyperlinked to other portions of this standard are shown in **underlined-blue text** (hyperlinking works when this standard is viewed interactively as a PDF file).
- In the formal language definitions, syntactic keywords and tokens are shown in **boldface-red text**.

1.8 Contents of this standard

The organization of the remainder of this standard is as follows:

- [Clause 2](#) provides references to other applicable standards that are assumed or required for this standard.
- [Clause 3](#) defines terms, acronyms, and abbreviations used throughout the different specifications contained in this standard.
- [Clause 4](#) defines the interoperability use model.
- [Clause 5](#) defines the bus and abstraction interface definitions.
- [Clause 6](#) defines the component and interconnect models.
- [Clause 7](#) defines the designs and their connections.
- [Clause 8](#) defines the abstractor model between abstraction definitions.
- [Clause 9](#) defines the type definition descriptions.
- [Clause 10](#) defines the generator chain configuration.
- [Clause 11](#) defines design configurations.
- [Clause 12](#) defines the catalog feature.
- [Clause 13](#) defines addressing.
- [Clause 14](#) defines data visibility.
- [Annex A](#) Bibliography.
- [Annex B](#) Semantic consistency rules.

- [Annex C](#) Common elements and concepts.
- [Annex D](#) Types.
- [Annex E](#) SystemVerilog expressions.
- [Annex F](#) Tight generator interface.
- [Annex G](#) External bus with an internal/digital interface.
- [Annex H](#) Bridges and channels.
- [Annex I](#) Examples.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used; therefore, each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

eXtensible Markup Language (XML) schema specification, available from the W3C Web site at <http://www.w3.org/TR/xmlschema-0/>; <http://www.w3.org/TR/xmlschema-1/>; <http://www.w3.org/TR/xmlschema-2/>.

eXtensible Markup Language (XML) version 1.0 specification, available from the W3C Web site at <http://www.w3.org/TR/REC-xml/>.

eXtensible Stylesheet Language Transformations (XSLT) version 1.0 specification, available from the W3C Web site at <http://www.w3.org/TR/xslt>.

IEEE Std 1800™, IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language.^{21, 22}

REpresentational State Transfer (REST), Web Services Architecture, description of relationship to World Wide Web on the W3C Web site at <https://www.w3.org/TR/ws-arch/#relwwwrest>

Simple Object Access Protocol (SOAP) version 1.2 specification, available from the W3C Web site at <http://www.w3.org/TR/soap12-part1/>.

Web Services Description Language (WSDL) version 1.1 specification, available from the W3C Web site at <http://www.w3.org/TR/wsdl>.

XML schema NameChar definition, available from the W3C Web site at <http://www.w3.org/TR/REC-xml/#NT-NameChar>.

XML schema NameStartChar definition, available from the W3C Web site at <http://www.w3.org/TR/REC-xml/#NT-NameStartChar>.

²¹IEEE publications are available from The Institute of Electrical and Electronics Engineers, Inc. (<http://standards.ieee.org/>).

²²The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

3. Definitions, acronyms, and abbreviations

For the purposes of this document, the following terms and definitions apply. *The IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.²³

3.1 Definitions

abstraction definition: An object that describes a representation of a bus interface, including details of the ports this type of bus interface may have and the constraints that apply to these ports.

abstractor: A top-level IP-XACT element used to convert between two bus interfaces having different abstraction types and sharing the same bus type.

active interface: An interface that participates in the transactions.

ad hoc connection: A direct connection between two or more ports without the use of bus interfaces.

application programmers interface (API): A method for accessing design data and meta-data in a procedural way.

bridge: A mechanism to model the internal relationship between initiator interfaces and target interfaces inside a component. Bridges explicitly describe the internal point-to-point connections between the component interfaces. A bridge can have multiple address spaces, supports memory mapping and remapping, and can have only direct interfaces. *Syn: bus bridge.*

broadcast: Depending upon the context, the term *broadcast* may have several meanings. First, it may be interface connection type where one interface connects to multiple other interfaces. Secondly, when used in the context of register fields, broadcast may mean that a write action to a register field also implies a write action to other register fields. This is indicated by the broadcast element inside a field.

bus: A collection of ports used to connect blocks connected to it involving both hardware and software protocols.

bus definition: An object that describes the type properties for a bus, such as the maximum initiators allowed or if one bus expands upon the definition of another.

bus interface: The interface of an intellectual property (IP) to an interconnection. Components are connected together by linking the bus interfaces together. The three different classes of bus interfaces, initiator, target, and system, have two flavors: direct and mirrored.

catalog: An object that defines a mapping from IP-XACT Vendor Library Name Version (VLNV) to the physical file in which the IP-XACT object is defined.

channel: A mechanism to model the internal relationship between mirrored-target interfaces and mirrored-initiator interfaces inside a component. A channel can also represent a simple wiring interconnection or a more complex structure, such as a bus. A channel can have only one address space. Channel interfaces are always mirrored interfaces. A channel supports memory mapping and remapping.

component: An object containing IP-XACT meta-data for an intellectual property (IP). Components are used to describe computing cores, peripherals, and bussing structures. *Syn: component description.*

²³*The IEEE Standards Dictionary Online* subscription is available at
http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.

configurable element: An element in an IP-XACT description that can be set to a new value by a user, generator, or dependency equation. This includes all elements with a resolve attribute of user or generated.

configurable intellectual property (IP): IP that contains configurable elements and/or an IP-specific generator capable of creating new components from the configured component and updating the design with the new version of the component. *Syn:* **configurable component**.

configuration: Replacing the parameter values with values specified in configurableElementValue elements.

NOTE—See [C.18](#).

constraint: A limitation on a part of the system that needs to be satisfied for the system to be correct. Timing constraints are often specified on ports, requiring that during a given clock cycle, the value of the port becomes stable in a certain time period and remains stable for a certain time period relative to a particular clock edge.

constraint set: Constraints defined in groups to associate different constraints with different views of the component.

design: An IP-XACT description of a system or subsystem listing its components and the connections between these components.

design configuration: A design description that contains non-essential ancillary information for generators, the active or current view selected for instances in the design, and configurable information defined in vendor extensions. It references a design description, can specify a view for the component instances and abstractors for each interconnection, and can configure generator chains. *Syn:* **configuration**.

design database: Working storage for both meta-data and component information that helps create and verify systems and subsystems.

design environment (DE): A software tool or set of tools that manages the access to IP-XACT objects, coordinates the execution of generator chains, and provides an application programmers interface (API) between generators and IP-XACT objects.

electronic design intellectual property (IP): In the electronic design community, a reusable collection of design specifications that represent the behavior, properties, and/or description of the design in various media. The name IP is partially derived from the common practice of considering a collection of this type to be the intellectual property of one party. Both hardware and software collections are encompassed by this term. IP utilized in the context of a system on chip (SoC) design or design flow may include specifications; design models; design implementation descriptions; verification coordinators, stimulus generators, checkers, and assertion/constraint descriptions; soft design objects (such as embedded software and real-time operating systems); and design and verification flow information and scripts. IP-XACT distinguishes between fixed IP and configurable IP.

endianness: The system of ordering bytes in computer memory where big endian is the most significant byte at the lowest memory address and little endian is the least significant byte at the lowest memory address.

eXtensible Markup Language (XML): A simple, very flexible text format derived from Standard Generalized Markup Language (SGML). See ISO/IEC 8879.

eXtensible Stylesheet Language Transformations (XSLT): A language for transforming eXtensible Markup Language (XML) documents into other XML documents.

fixed intellectual property (IP): IP that has no elements that are configured by the design environment (DE) or set by industry de facto tools.

generator: An external application that can communicate with a design environment (DE) using the tight generator interface (TGI) to complete a specific task. Common examples include documentation generation, design data manipulation (adding/removing/updating IP-XACT elements), and flow automation.

generator chain: A hierarchical list of generators used to define the order for executing generators. A design flow can be represented by a generator chain.

generator group: A symbolic name assigned to a generator to enable generator selection.

generator invocation: A method of running an application at a defined phase in the generator group with a given number of elements.

hardware description language (HDL) path: A hierarchical path to a memory mapped object (register or memory) within a design. HDL paths facilitate back-door accesses. In IP-XACT, a view-specific HDL path can be defined for each memory mapped object. HDL paths are stored in a distributed manner across the memory map hierarchy using an accessHandle.

NOTE—See [C.1](#).

hierarchical bus interface: A hierarchical bus interface is a bus interface of an hierarchical component.

hierarchical component: A component that has one or more **views** that reference IP-XACT design or design configuration descriptions.

hierarchical family of bus interfaces: A hierarchical family of bus interfaces is a set of bus interfaces composed of a hierarchical bus interface and all its descendants.

hierarchical family of components: A hierarchical family of components is a component and all its hierarchical descendants.

indirect interface: The access mechanism for an indirectly accessible memory map. This includes defining indirect address and data fields along with addressing information.

indirectly accessible memory map: A memory map whose contents cannot be accessed at a fixed address via a bus interface and, instead, are indirectly accessible via indirect address and data fields. A bus interface accesses an indirectly accessible memory map via reads and writes to indirect address and data fields.

initiative: An abstract description of port modes (requires, provides, both, or phantom) used for transaction-level modeling (TLM).

initiator interface: The bus interface that initiates a transaction (like a read or write) on a bus.

intellectual property (IP) integrator: A party in the design process who receives IP and subsystems and combines them into a larger system.

intellectual property (IP) platform architect: The creator of platform-based architectures.

intellectual property (IP) provider: The creator and supplier of IP.

intellectual property (IP) repository: The database of IP.

interconnection: The connection between two or more bus interfaces.

leaf component: A component that does not contain any views that reference IP-XACT design or design configuration descriptions.

memory map: A block of memory in a component (which may be accessible through a target interface).

meta-data: A tool-interpretable way of describing information, such as the design history, locality, association, configuration options, and integration requirements of an object as well as constraints against an object.

mirror interface: An interface that has the same (or similar) ports as its related direct bus interface, but whose port directions are reversed. Therefore, a port that is an input on a direct bus interface would be an output in the matching mirror interface.

monitor interface: An interface used in verification that is not a initiator, target, or system interface.

multi-layer buses: Buses that have to be modeled as component bridges with direct interfaces or as a hierarchical component.

objects: XML descriptions of the following types: catalogs, components, designs, busDefinitions, abstractionDefinitions, abstractors, designConfigurations, and generatorChains. To be able to be uniquely referenced, each object has an unique identifier called its Vendor Library Name Version (VNV).

opaque bridge: A bus interconnect component that may modify the address space of a initiator bus interface of one bus type to the memory map of a target bus interface of another bus type and does not allow direct access to any components residing on that address space.

phantom port: A port that exists only in an IP-XACT component; it does not exist on any hardware description language (HDL) or transaction-level modeling (TLM) component.

phase number: The sequence in which generators should be fired.

platform: Architectural (sub)system framework.

platform consumer: The user/group that builds a system on chip (SoC) based on a particular platform.

platform provider: The user/group that develops and delivers platforms to platform consumers.

platform rules: Rules that define how components interface with a specific platform.

port: The interface items of a component. These interface items allow dynamic exchange of information. Connections between ports may be specified by using ad hoc connections or by including them in bus interfaces connected together by interconnections.

schema: A means for defining the structure, content, and semantics of eXtensible Markup Language (XML) documents.

segment: A portion of an addressSpace, defined with an address offset and range.

semantic consistency rules (SCRs): Additional rules applied to an eXtensible Markup Language (XML) description that cannot be expressed in the schema. Typically, these are rules between elements in multiple XML descriptions.

structured port: A port that has sub-ports organized as a structure, union, or System Verilog interface. The leaf ports of a structured port are wire ports.

system interface: An interface that is neither a initiator nor target interface and allows specialized (or non-standard) connections to a bus (e.g., clock).

target interface: The bus interface that terminates or consumes a transaction initiated by a initiator interface. Target interfaces often contain information about the registers accessible through the target interface.

tight generator interface (TGI): An interface used to manipulate values of elements and attributes for IP-XACT-compliant XML.

transactional port: A port that has an initiative and a kind or a type definition (which can specify the data type of the port). Transactional ports are used for high-level modeling.

transaction-level modeling (TLM): An abstraction level higher than register transfer level (RTL), used for specifying, simulating, verifying, implementing, and evaluating system on chip (SoC) designs.

transparent bridge: A bus interconnect component that modifies the address space of a initiator bus interface of one bus type to the memory map of a target bus interface of another bus type with directly addressable access to any components residing on that address space.

type definitions: a top-level IP-XACT document describing memory-related types.

use model: A process method of working with a tool.

user interface: Methods of interacting between a tool and its user.

validation: The process of proving the correctness of construction of a set of components.

Vendor Library Name Version (VLNV): The unique identifier assigned to each top-level IP-XACT object and specified in each eXtensible Markup Language (XML) file.

verification: The process of proving the behavior of a set of connected components.

verification intellectual property (VIP): Components included in a design for verification purposes.

view: An implementation of a component. A component may have multiple views, each with its own function in the design flow.

virtual register: A collection of fields, overlaid on top of a memory, usually in an array. The semantics and layout of virtual registers come from an agreement between the software and hardware. Virtual registers are modeled in IP-XACT by creating register instances within addressBlocks with usage memory. The child elements of a virtual register are restricted to a subset of the overall register element.

NOTE—See [6.12.4](#).

virtual register file: A grouping of virtual registers. A virtual register file is modeled using the registerFile element in an addressBlock element with usage of memory. The children of virtual register files shall be virtual.

NOTE—See [6.12.4](#).

wire connections: Connections that connect wire ports, wire sub-ports, or structured ports.

wire port: A port that describes binary values or an array of binary values. Wire ports can have a direction: in, out, or inout.

3.2 Acronyms and abbreviations

API	application programmers interface
DE	design environment
EDA	electronic design automation
HDL	hardware description language
HTTP	Hypertext Transfer Protocol
IP	(electronic design) intellectual property
LAU	least addressable unit (of memory)
RAM	random access memory
REST	REpresentational State Transfer
ROM	read-only memory
RTL	register transfer level (design)
SCR	semantic consistency rule
SOAP	Simple Object Access Protocol
SoC	system on chip
TGI	tight generator interface
TLM	transaction-level modeling
VIP	verification IP
VLNV	Vendor Library Name Version
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformations

4. Interoperability use model

To introduce the use model for the IP-XACT specifications, it is first necessary to identify specific roles and responsibilities within the model and then relate these to how the IP-XACT specifications impact their interactions. All or some of the roles can be mixed within a single organization, e.g., some EDA providers are also providing IP, a component IP provider can also be a platform provider, and an IP system design provider may also be a consumer.

4.1 Roles and responsibilities

For this standard, the roles and responsibilities are restricted to the scope of IP-XACT HDL and TLM system design.

4.1.1 Component IP provider

A component IP provider is a person, group, or company creating IP components or subsystems for integration into a SoC design. These IPs can be hardware components (e.g., processors, memories, buses), verification components, and/or hardware-dependent software elements. They may be provided as source files or in a compiled form (i.e., simulation model). An IP is usually provided with a functional description, a timing description, some implementation or verification constraints, and some parameters to characterize (or configure) the IP. All these types of characterization data may be described as meta-data compliant with the IP-XACT schema. Elements not already provided in the base schema can be provided using namespace extensibility mechanisms of the specification.

The IP provider can use one or more EDA tools to create/refine/debug IP. During this process, the IP provider may export and re-import its design from one environment to another. The IP-XACT IP descriptions need to enable this exchange for component IP.

At some point, this IP can be transferred to customers, partners, and external EDA tool suppliers by using IP-XACT-compliant XML. IP can be characterized as *fixed IP* or *configurable IP*.

4.1.2 SoC design IP provider

A SoC design IP provider is a person, group, or company that integrates and validates IP provided by one or more IP providers to build system platforms, which are complete and validated systems or subsystems. Like the IP provider, the platform provider can use EDA tools to create/refine/debug its platform, but at some point the IP needs to be exchanged with others (e.g., customers, partners, other EDA tools). To do so, the platform IP has to be expressed in the IP-XACT-specified format as a hierarchical component.

4.1.3 SoC design IP consumer

A SoC design IP consumer is a person, group, or company that configures and generates system applications based on platforms supplied by SoC design IP providers. These platforms are complete system designs or subsystems. Like the platform provider, the platform consumer can use EDA tools to create/refine/debug its system application and/or configure the design architecture. To do so, the EDA tool needs to support any platform IP expressed in the IP-XACT-specified format.

4.1.4 Design tool supplier

A design tool supplier is a group or company that provides tools to verify and/or implement an IP or platform IP. There are three major tools (which could be combined) provided in a system flow:

- Platform builder (or *system design environment*) tools: These help to assemble a platform with some automation (e.g., automatic generation of interconnect).
- Verification point tools: These handle functional and timing simulation, verification, analysis, debugging, co-simulation, co-verification, and acceleration.
- Implementation point tools: These handle synthesizing, floor-planing, place, routing, etc.

The EDA provider needs to be able to import IP-XACT component or system IP libraries from multiple sources and export them in the same format.

Further, IP-XACT EDA tools need to recognize, associate, and launch generators that may be provided by a generator or IP provider in support of configurable IP bundles. The imported IP might need to be created and/or modified by the tool and then exported back (e.g., to be exchanged with other EDA vendor tools) to satisfy the customer design flow.

To further support any generators supplied with IP bundles, the IP-XACT DE tools need to be able to recognize and interface with generator-wrapped point tools. These may be provided by another EDA provider or by the IP designer/consumer as part of a company's internal design and verification flow. In general, these support specialized design-automation features, such as architectural-rule checking.

4.2 IP-XACT IP exchange flows

This subclause describes a typical IP exchange flow that the IP-XACT specifications technically support between the roles defined in 4.1. By way of example, the following specific exchange flow can benefit from use of the IP-XACT specification:

The component IP provider generates an IP-XACT XML package and sends it to a SoC design tool (EDA tool supplier) or directly to a platform (i.e., SoC design IP) provider. The EDA tool supplier imports IP-XACT XML IP and generates platform IP and/or updates (configures) the IP components. The platform provider generates a configurable platform IP and exports it in IP-XACT XML format, which the end user imports to build system applications. The platform provider can also generate its own platform IP into IP-XACT format and send it to the EDA provider.

Although many different possible IP exchange flows exist, from the user's viewpoint, there are three main use models, as follows:

- IP (component or SoC design) provider use model
- Generator (IP provider and design tool) provider use model
- SoC design tool provider use model

4.2.1 Component or SoC design IP provider use model

The IP provider (a hardware component IP designer or platform IP architect) can use IP-XACT to package IP in a standard and reusable format. The first step consists in creating an IP-XACT XML package (XML plus any IP views) to export the IP database in a valid format. To express this IP as an IP-XACT IP, the IP provider needs to parse the entire design description tree (which is composed of files of different types: HDL source files, data sheets, interfaces, parameters, etc.) and package it into an IP-XACT XML format. This can

be a manual step (by directly editing IP-XACT-compliant XML) or an automated one (using scripts to generate schema-compliant IP-XACT XML).

Once the IP has been packaged in an IP-XACT format, the IP provider can use a SoC design tool to write/debug/simulate/implement the IP.

4.2.2 Generator provider use model

The author of a generator expects to interact with the SoC design tool through a fixed interface during well-defined times in the design life cycle: when components are instantiated or modified or when a generator chain is started.

Generators are used within the SoC design tool to extend its capabilities: wrapping a point tool, e.g., a simulator; wiring up IP within the design; or checking the design is correct or maybe modifying the design. Many of these features may be supplied by the IP author and handled by generators embedded in the IP itself.

Consequently, there are at least two groups of generator providers: IP vendors who supply generators that are written specifically to support their IP and generic generator authors who wish to extend the features available within the SoC design tool. This latter group will be mainly SoC design tool vendors at first, but will also come to include third-party generator vendors.

4.2.3 System design tool provider use model

The system design tool takes IP-XACT components and designs as input, configures them, and loads them into its own database format. Then it can automate some tasks, such as creating the platform, generating the component interconnect and bus fabric, and generating or updating the IP-XACT IP as an output (by providing new or updated XML with the attached information, e.g., new source files, parameters, documentation).

Customer design flows are usually composed of a chain of different tools from the same or different EDA vendors (e.g., when an EDA provider is not providing the entire tool chain to cover all the user flow or the customer is selecting the best-in-class point tools). To address this requirement, the EDA vendor providing an IP-XACT-enabled tool needs to read and produce the IP-XACT-specified format and then utilize and implement the interfaces defined by IP-XACT documents. In this use model, each SoC design tool uses its own generators (possibly utilizing the IP-XACT TGI) to build and update its internal meta-data state and export to an IP-XACT format. Then the IP-XACT description can be imported by another IP-XACT-enabled EDA tool.

5. Interface definition descriptions

5.1 Definition descriptions

In IP-XACT, a group of ports that together perform a function are described by a set of elements and attributes split across two descriptions: a bus definition and an abstraction definition. These two descriptions are referenced by components or abstractors in their bus or abstractor interfaces.

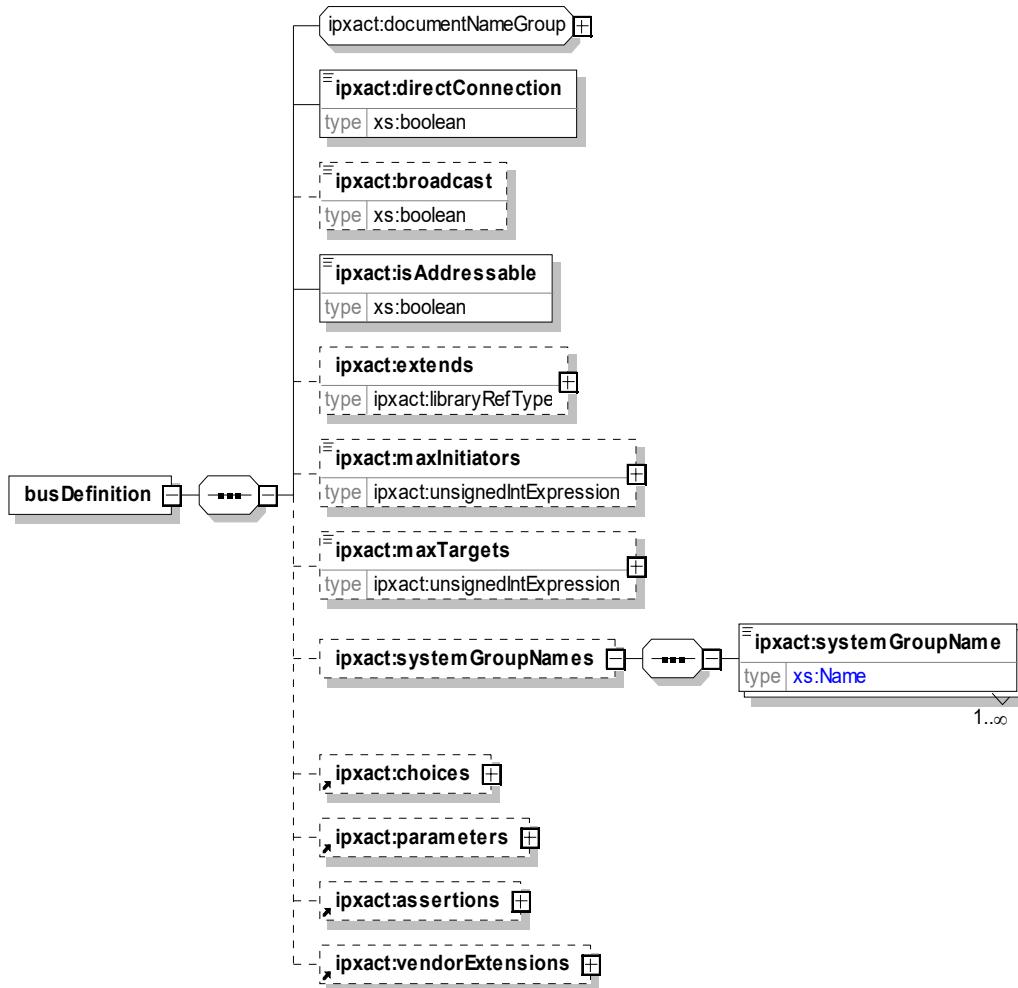
The *bus definition* description contains the high-level attributes of the interface, including items such as the connection method and indication of addressing.

The *abstraction definition* contains the low-level attributes of the interface, including items such as the name, direction, and width of the ports. This is a list of logical ports that may appear on a bus interface for that bus type. Multiple abstractions definitions can be associated with a single bus definition. See [6.7](#).

5.2 Bus definition

5.2.1 Schema

The following schema details the information contained in the **busDefinition** element, which is one of the top-level elements in the IP-XACT specification used to describe the high-level aspects of a bus.



5.2.2 Description

The top-level **busDefinition** element describes the high-level aspects of a bus or interconnect. It contains the following elements and attributes:

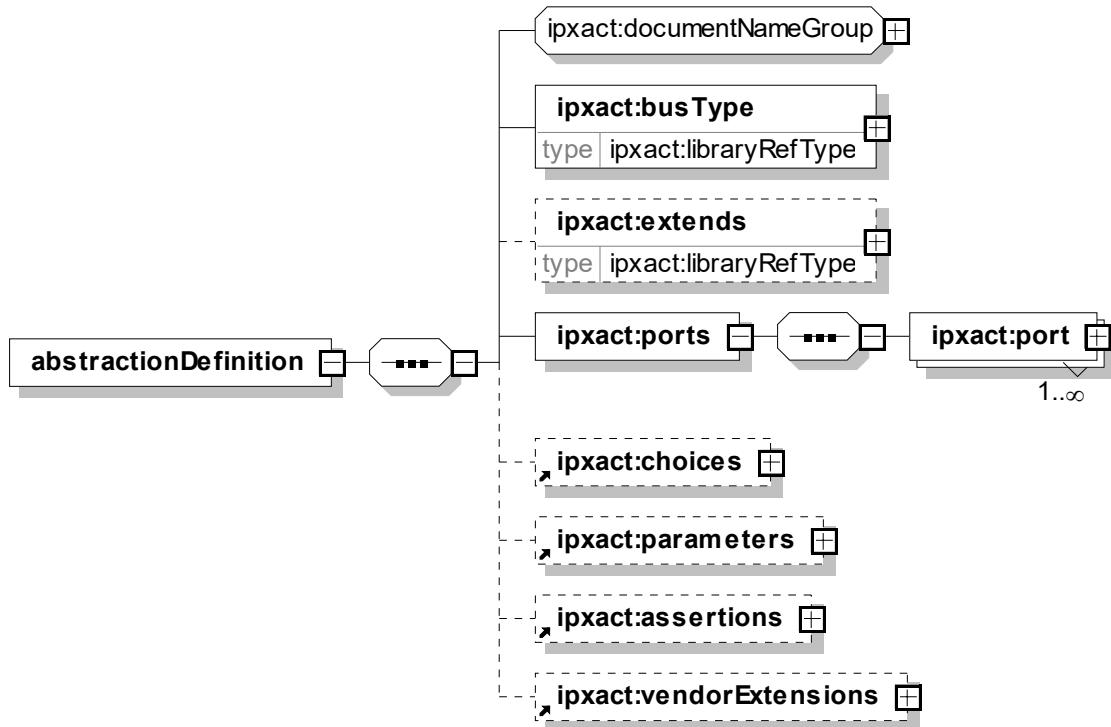
- a) The **documentNameGroup** contains **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. See [C.28](#).
- b) **directConnection** (mandatory; type: **boolean**) specifies what connections are allowed. A value of **true** specifies these interfaces may be connected in a direct initiator-to-target fashion. A value of **false** indicates only non-mirror to mirror type connections are allowed (i.e., initiator-mirroredInitiator, target-mirroredTarget, or system-mirroredSystem).
- c) **broadcast** (optional; type: **boolean**; default: **false**) indicates this bus definition supports *broadcast mode*, i.e., bus interfaces using this definition support one-to-many interface connections.
- d) **isAddressable** (mandatory; type: **boolean**) specifies the bus has addressing information. A value of **true** specifies these interfaces contain addressing information and a memory map can be traced through this interface. A value of **false** indicates these interfaces do not contain any traceable addressing information. See also [6.5](#).
- e) **extends** (optional; type: **libraryRefType**, see [C.18](#)) specifies whether this definition is an extension from another bus definition. See also [5.12](#).
- f) **maxInitiators** (optional; type: **unsignedIntExpression**, see [C.7.11](#)) specifies the maximum number of initiators that can appear in a **channel** element containing bus interfaces using this bus definition. If the **maxInitiators** element is not present, the number of allowed initiators is unbounded.
- g) **maxTarget** (optional; type: **unsignedIntExpression**, see [C.7.11](#)) specifies the maximum number of targets that can appear in a **channel** element containing bus interfaces using this bus definition. If the **maxTargets** element is not present, the number of allowed targets is unbounded.
- h) **systemGroupNames** (optional) defines an unbounded list of **systemGroupName** (mandatory; type: **Name**) elements, which in turn define the possible group names to be used under an **onSystem** element in an abstraction definition. The definition of the group names in the bus definition allows multiple abstraction definitions to indicate which system interfaces match each other. The **systemGroupName** shall be unique with the containing **systemGroupNames** element.
- i) **choices** (optional) specifies multiple enumerated lists. These lists are referenced by other sections of this bus definition description. See [C.21](#).
- j) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this bus definition. See [C.21](#).
- k) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.5](#).
- l) **vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See [C.27](#).

See also [SCR 1.3](#), [SCR 1.10](#), [SCR 1.12](#), [SCR 2.17](#), [SCR 3.18](#), and [SCR 6.20](#).

5.3 Abstraction definition

5.3.1 Schema

The following schema details the information contained in the **abstractionDefinition** element, which is one of the top-level elements in the IP-XACT specification used to describe the low-level aspects of a bus.



5.3.2 Description

The **abstractionDefinition** element describes the low-level aspects of a bus or interconnect. It contains the following elements and attributes:

- The **documentNameGroup** contains **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. See [C.28](#).
- busType** (mandatory; type: **libraryRefType**, see [C.18](#)) specifies the bus definition that this abstraction defines. See also [5.12](#).
- extends** (optional; type: **libraryRefType**, see [C.18](#)) specifies whether this definition is an extension from another abstraction definition. The extending abstraction definition may change the definition of logical ports, add new ports, or mark existing logical ports illegal (to disallow their use). See [5.12](#).
- ports** (mandatory) is a list of logical ports. See [5.4](#).
- choices** (optional) specifies multiple enumerated lists. These lists are referenced by other sections of this abstraction definition description. See [C.21](#).
- parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this bus definition. See [C.21](#).
- assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.5](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See [C.27](#).

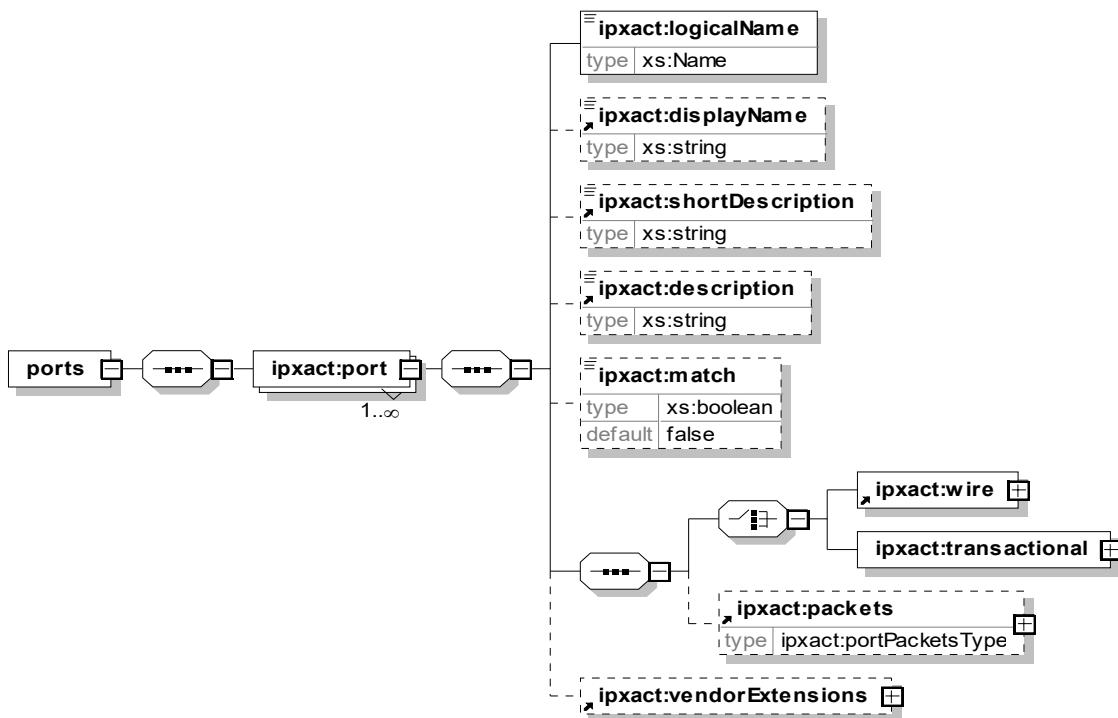
The **abstractionDefinition** element contains a list of logical ports that define a representation of the bus type to which it refers. A port can be a wire port (see [5.7](#)), or a transactional port (see [5.9](#)). A *wire port* carries logic information or an array of logic information. A *transactional port* carries information that is represented at a higher level of abstraction.

See also [SCR 1.10](#), [SCR 1.12](#), [SCR 1.14](#), [SCR 3.1](#), [SCR 3.15](#), [SCR 3.16](#), and [SCR 6.11](#).

5.4 Ports

5.4.1 Schema

The following schema details the information contained in the **ports** element, which appears as part of the **abstractionDefinition** element within an abstraction definition. This is different from the **ports** element that appears as part of the **model** element within components.



5.4.2 Description

A **port** element defines the logical port information for the containing abstraction definition. It contains the following elements:

- logicalName** (mandatory; type: *Name*) gives a name to the logical port that can be used later in component description when the mapping is done from a logical abstraction definition port to the components physical port. The **logicalName** shall be unique within the **abstractionDefinition**.
- displayName** (optional; type: *string*) allows a short descriptive text to be associated with the port. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- shortDescription** (optional; type: *string*) is a more compact textual description of the containing element compared to description.

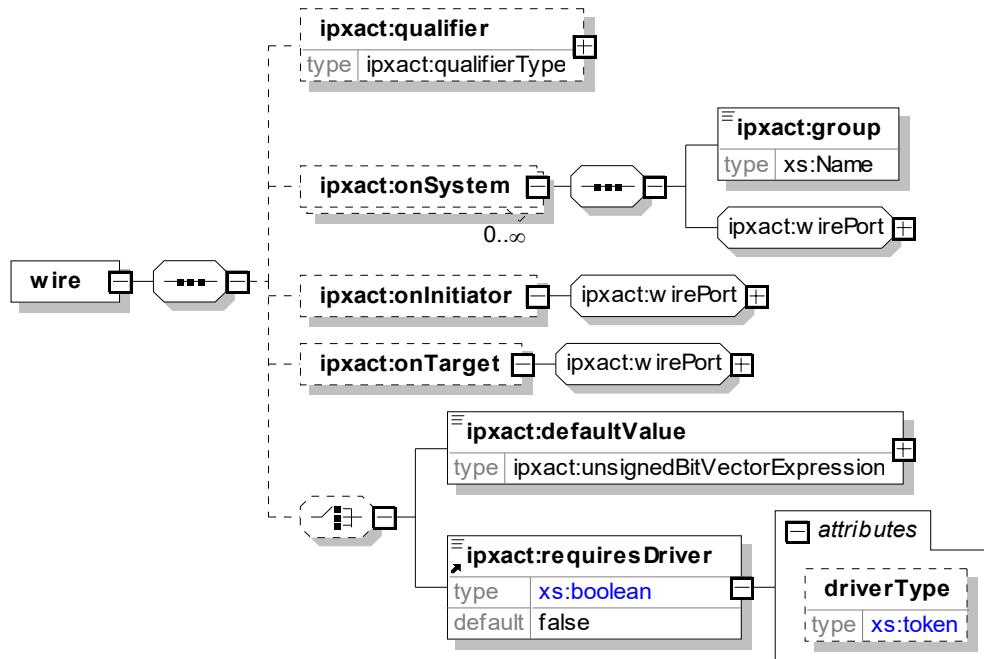
- d) **description** (optional; type: *string*) allows a textual description of the port.
- e) **match** (optional; type: *boolean*) When match has a value of **true**, then the ports in a connection need to be present on both sides of the connection. For example, if the port is mapped on one side of the connection then the port shall be mapped on the other side of the connection as well. The width of this port on a connection should match the width of other ports on a connection. The default is **false**.
- f) Each **port** also requires a **wire** element or a **transactional** element to further describe the details about this port. See [5.5](#) or [5.9](#), respectively. A **wire** style port is a port that carries logic values or an array of logic values. A **transactional** style port is a port that carries any other type of information, typically used for TLM.
- g) A port may contain zero or one **packets** and packets contains one or more packet elements. Each packet contains: nameGroup, endianness, packetFields, and vendorExtension. See [5.11](#).
- h) **vendorExtensions** (optional) contains any extra vendor-specific data related to the port. See [C.27](#).

See also [SCR 6.33](#).

5.5 Wire ports

5.5.1 Schema

The following schema details the information contained in the **wire** element, which may appear as part of the **port** element within an abstraction definition (**abstractionDefinition/ports/port**).



5.5.2 Description

A **wire** element represents a port that carries logic values or an array of logic values. This logical wire port may provide optional constraints for a wire port, to which it is mapped inside a component or abstractor's **busInterface**. It contains the following elements and attributes:

- a) **qualifier** (optional) indicates which type of information this wire port carries. See [5.6](#).
- b) **onSystem** (optional) defines constraints, e.g., timing constraints, for this wire port if it is present in a system bus interface with a matching group name.
 - 1) The **group** (mandatory) attribute indicates the group name for the wire port. It distinguishes between different sets of system interfaces. Usually, all the arbiter ports are processed together, or all the clock or reset ports are processed together. So, this is really a mechanism to specify any sort of non-standard bus interface capabilities for the interconnect. The type of this element is *Name*.
 - 2) The group **wirePort** specifies what elements are used in this port. See [5.7](#).
- c) **onInitiator** (optional) defines constraints for this wire port when present in an initiator bus interface. The group **wirePort** specifies what elements are used in this port. See [5.7](#).
- d) **onTarget** (optional) defines constraints for this wire port when present in a target bus interface. The group **wirePort** specifies what elements are used in this port. See [5.7](#).
- e) A wire may also contain one of the following elements:
 - 1) **defaultValue** (type: *unsignedBitVectorExpression*, see [C.7.10](#)) contains the default logic value for this wire port. See [C.31](#) to determine when this value is applied.
 - 2) **requiresDriver** (type: *boolean*; default: **false**) specifies whether the port requires a driver when used in a completed design. If this element is not present, its effective value is **false**, indicating this port does not require a driver. When set to **true**, the attribute **driverType** further qualifies what driver type is required: **any** (any logic signal or value), **clock** (a repeating type waveform), or **singleshot** (a non-repeating type waveform). If **driverType** is not present, its effective value is **any**.

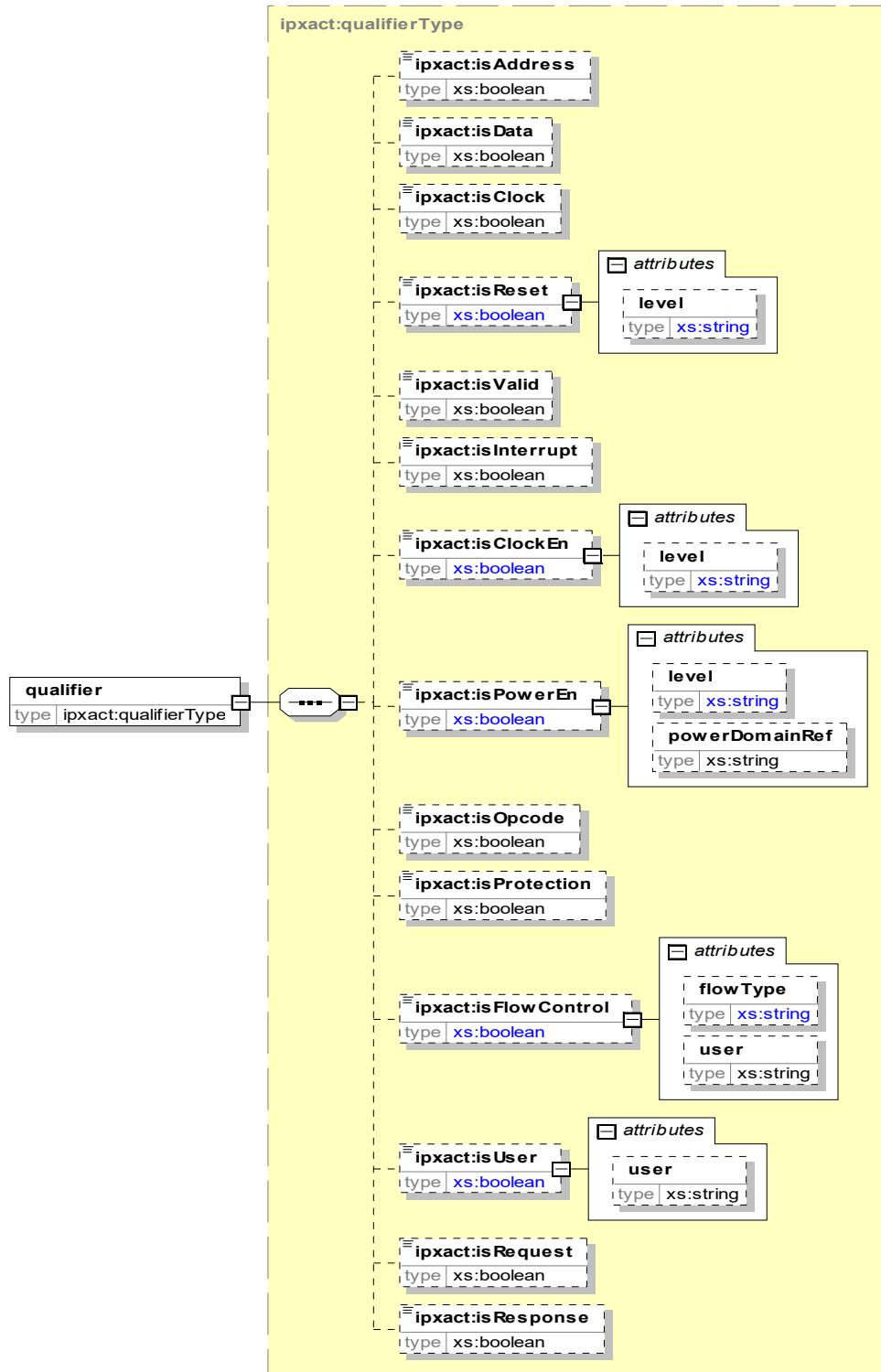
NOTE—The **onInitiator**, **onTarget**, and **onSystem** elements associated with each logical port provide optional constraints. If any of these is missing, there are no constraints for how the port appears on interfaces with that mode (initiator, system, or target). A port may appear in any system interface group unless its presence is marked as illegal for that group. The abstraction definition author has the choice of how far to constrain the definitions. Generally speaking, more constraints in the definitions reduce implementation flexibility for whoever is creating IP with interface that conforms to the abstraction definition.

See also [SCR 6.12](#) and [SCR 6.14](#).

5.6 Qualifiers

5.6.1 Schema

The following schema details the information contained in the **qualifier** element, which may appear as part of the **wire** element within an abstraction definition (**abstractionDefinition/ports/port/wire**).



5.6.2 Description

The **qualifier** element indicates which type of information a wire port carries. It contains the following elements:

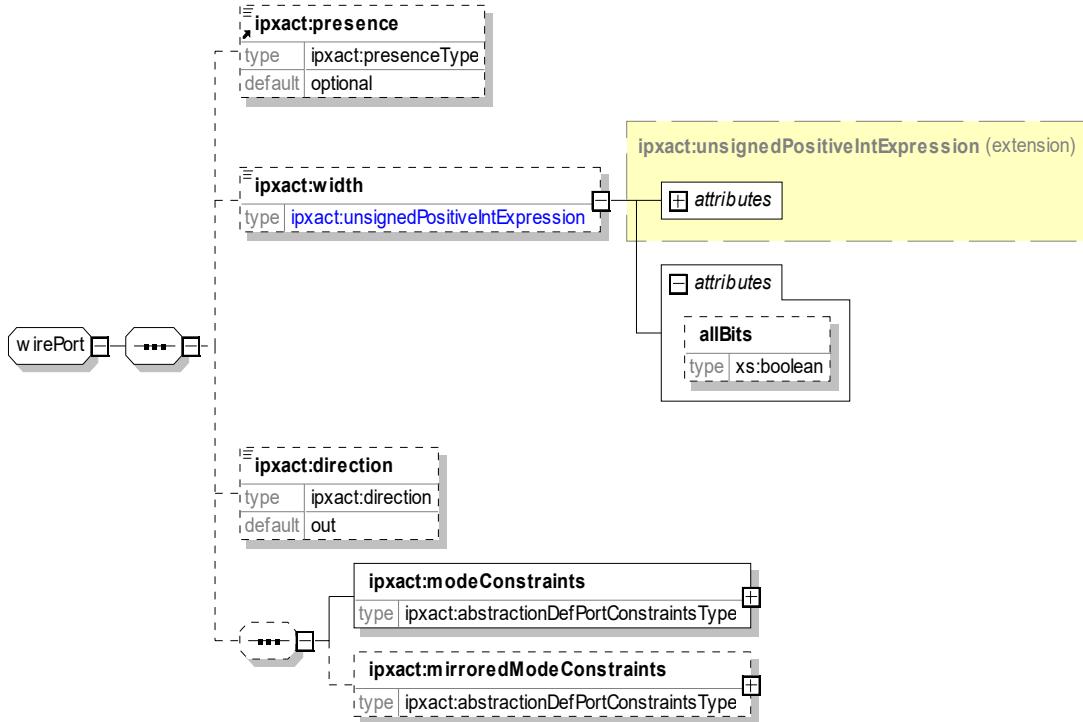
- a) **isAddress** (optional; type: *boolean*) when **true**, specifies the port contains address information. See also [Clause 14](#).
- b) **isData** (optional; type: *boolean*) when **true**, specifies the port contains data information. This data resides in registers defined in the memory map referenced by the interface. The width defined by the port on each side of the two connected bus interfaces can be used to determine which portions of the data may be lost or gained (tied off to defaults) during transfers if the two widths do not match. See also [Clause 14](#).
- c) **isClock** (optional; type: *boolean*) when **true**, specifies this port is a clock for this bus interface, i.e., it provides a repeating pattern that the interface uses to implement the protocol. No method of processing is implied with this tag.
- d) **isReset** (optional; type: *boolean*) when **true**, specifies this port is a reset for this bus interface., i.e., it provides the necessary input to put the interface into a known state. The **level** attribute to this tag that can be “low” or “high” and is used to document the assertion level for the reset signal. If no level attribute is present, then no assumption can be made on the assertion level.
- e) **isValid** (optional; type: *boolean*) specifies the data on the interface is currently valid. No method of processing is implied with this tag.
- f) **isInterrupt** (optional; type: *boolean*) specifies the port contains interrupt information. No method of processing is implied with this tag.
- g) **isClockEn** (optional; type: *boolean*) specifies the port contains clock enable information. **isClockEn** has a **level** attribute similar to **isReset**. The **level** attribute describes the assertion level of the clock enable information. Possible values are low and high. If no **level** attribute is present, then no assumption can be made on the assertion level. No method of processing is implied with this tag.
- h) **isPowerEn** (optional; type: *boolean*) specifies the port contains power enable information. The **level** attribute describes the assertion level of the power enable information. Possible values are low and high. If no **level** attribute is present, then no assumption can be made on the assertion level. The **powerDomainRef** attribute references a power domain of the encapsulating component. It does not apply to abstraction definitions. See [6.3](#).
- i) **isOpcode** (optional; type: *boolean*) specifies the port contains operational code or opcode information. No method of processing is implied with this tag.
- j) **isProtection** (optional; type: *boolean*) specifies that the signal is used by a protection or security mechanism. It specifies the port contains protection or security information. No method of processing is implied with this tag.
- k) **isFlowControl** (optional; type: *boolean*) specifies that the port contains flow control information. The **flowType** attribute describes additional information on the type of flow control. Possible values are **credit-return**, **ready**, **busy**, and **user**. **creditReturn** indicates a credit is being returned. **ready** indicates the receiver is ready for a transaction. **busy** indicates that the receiver is busy and cannot accept an incoming message. The **user** attribute can be used to provide user-defined flow control types. No method of processing is implied with this tag.
- l) **isUser** (optional; type: *boolean*) specifies the port contains user-defined information. The **user** attribute can be used to describe user-defined information. No method of processing is implied with this tag.
- m) **isRequest** (optional; type: *boolean*) specifies the port contains request information. No method of processing is implied with this tag.
- n) **isResponse** (optional; type: *boolean*) specifies the port contains response information. No method of processing is implied with this tag.

See also [SCR 1.41](#) and [SCR 12.8](#).

5.7 Wire port group

5.7.1 Schema

The following schema details the information contained in the **wirePort** group, which may appear as part of the **onSystem**, **onInitiator**, or **onTarget** element within a **wire** element within an abstraction definition (**abstractionDefinition/ports/port/wire/onmode**).



5.7.2 Description

The **wirePort** group specifies what elements are used in a **wire** port. It contains the following elements:

- presence** (optional; default: **optional**) provides the capability to require or forbid a port from appearing in a **busInterface**. The three possible values are **illegal**, **required**, or **optional**. If this element is not present, its effective value is **optional**.
- width** (optional; type: **unsignedPositiveIntExpression**, see [C.7.13](#)) represents the number of logical bits that are required to represent this port. When mapping to this logical port in a **busInterface**/**portmap**, the numbering shall start from 0 to width-1. If **width** is not specified, the component shall define the number of bits in this port, but the logical portmap numbering shall still start at 0. If necessary, logical bit 0 shall be the least significant bit.
- allBits** (optional; type: **boolean**, default **false**) **False** means any number of bits can be mapped. **True** means all bits in the width shall be mapped.
- direction** (optional; default: **out**) restricts the direction of the port relative to the non-mirrored interface (see [6.4](#)). The three possible values are **in**, **out**, or **inout**.
- Each **wirePort** group can also have a sequence of **modeConstraints** and **mirroredModeConstraints** specifying the default synthesis constraints associated with this logical port. The **modeConstraints** apply to this port if it appears in a non-mirrored **mode** bus interface (see [C.7.13](#)).

[5.8](#)). Any **mirroredModeConstraints** apply to this port if it appears in a mirrored-*mode* bus interface (see [5.8](#)).

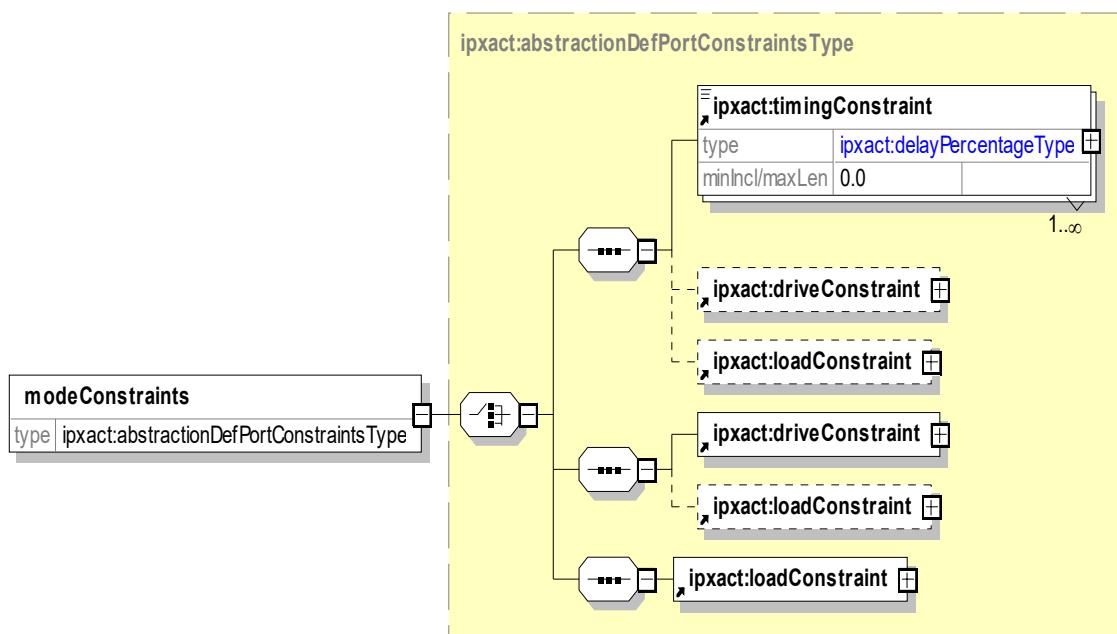
If **mirroredModeConstraints** are not specified, the **modeConstraints** also apply to this port in a mirrored-*mode* bus interface.

See also [SCR 6.5](#), [SCR 6.6](#), [SCR 6.7](#), and [SCR 6.16](#).

5.8 Wire port *mode* (and mirrored mode) constraints

5.8.1 Schema

The following schema defines the information contained in the **modeConstraints** and **mirroredModeConstraints** elements, which may appear within an **onInitiator**, **onTarget**, or **onSystem** element within an abstraction definition (**abstractionDefinition/ports/port/wire/onmode**).



5.8.2 Description

The **abstractionDefPortConstraintsType** type definition (used to define **modeConstraints** and **mirroredModeConstraints**) defines any default implementation constraints associated with the containing wire port of the abstraction definition. It contains one or more of the following elements:

- timingConstraint** (optional) element defines a technology-independent timing constraint associated with the containing wire port. See [6.15.16](#).
- driveConstraint** (optional) element defines a technology-independent drive constraint associated with the containing wire port. See [6.15.16](#).
- loadConstraint** (optional) element defines a technology-independent load constraint associated with the containing wire port. See [6.15.16](#).

The constraints contained within the **modeConstraints** element are applied to the corresponding physical ports in a component only when the physical port does not have any constraints defined within its own port

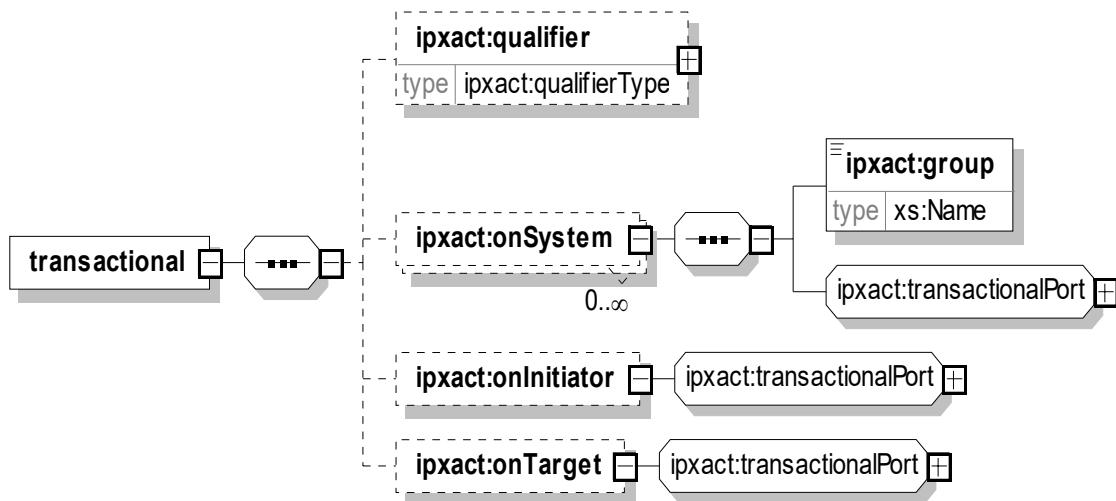
element and no design constraint file (fileType at SDC) is associated with the component in the current view.

For example, if it appears inside an **onInitiator** element, the constraints apply when the port appears in an initiator interface. If the **modeConstraints** element is immediately followed by a **mirroredModeConstraints** element, the constraints defined in the **modeConstraints** element apply only when the port is used in a non-mirrored *mode* interface and the constraints defined in the **mirroredModeConstraints** element are used when the port is used in a mirrored mode interface. Otherwise, the constraints apply when the port appears in a *mode* interface or a mirrored-*mode* interface.

5.9 Transactional ports

5.9.1 Schema

The following schema defines the information contained in the **transactional** element, which may appear within a **port** within an abstraction definition (**abstractionDefinition/ports/port**).



5.9.2 Description

The **transactional** element defines a logical transactional port of the abstraction definition. This logical transactional port may provide optional constraints for a transactional port, to which it is mapped inside a component or abstractor's **busInterface**. The **transactional** element also contains the following elements and attributes:

- The **qualifier** (optional) element indicates which type of information this transactional port carries. See [5.6](#).
- onSystem** (optional) defines constraints for this transactional port if it is present in a system bus interface with a matching group name.
 - The **group** (type: *Name*) attribute indicates the group name for the transactional port. It distinguishes between different sets of system interfaces. Usually, all the arbiter ports are processed together, or all the clock or reset ports are processed together. So, this is really a mechanism to specify any sort of non-standard bus interface capabilities for the interconnect. The **group** name shall match the one specified in the bus definition **systemGroupName**.
 - The group **transactionalPort** specifies what elements are used in this port. See [5.10](#).

- c) **onInitiator** (optional) defines constraints for this transactional port when present in an initiator bus interface. The group *transactionalPort* specifies what elements are used in this port. See [5.10](#).
- d) **onTarget** (optional) defines constraints for this transactional port when present in a target bus interface. The group *transactionalPort* specifies what elements are used in this port. See [5.10](#).

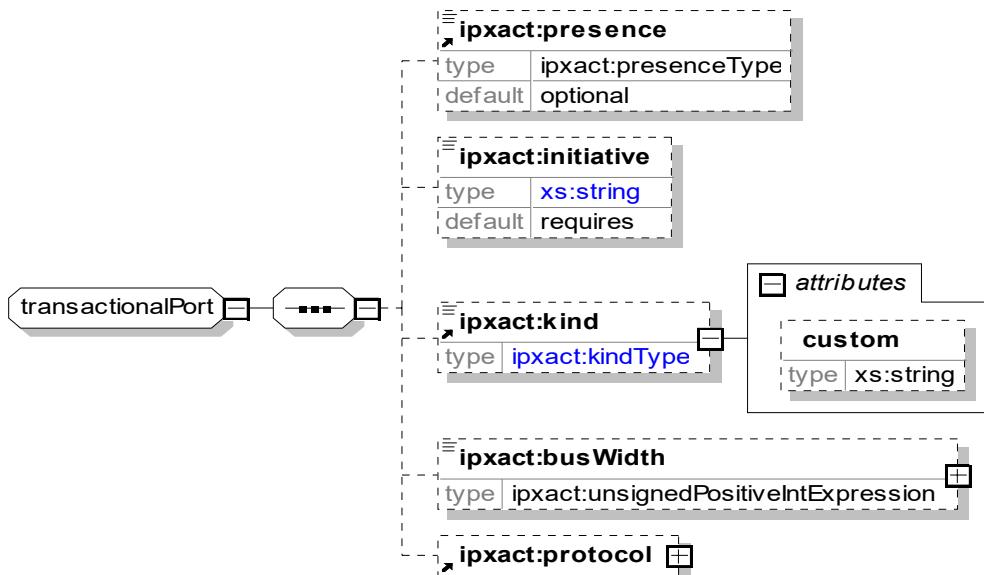
NOTE—The **onInitiator**, **onTarget**, and **onSystem** elements associated with each logical port provide optional constraints. If any of these is missing, there are no constraints for how the port appears on interfaces with that mode (initiator, system, or target). If no **onSystem** constraint is specified with a particular group, there are no constraints for system interfaces in that group. The abstraction definition author has the choice of how far to constrain the definitions. Generally speaking, more constraints in the definitions reduce implementation flexibility for whoever is creating IP with interface that conform to the abstraction definition.

See also [SCR 6.13](#), [SCR 6.14](#), and [SCR 6.20](#).

5.10 Transactional port group

5.10.1 Schema

The following schema defines the information contained in the **transactionalPort** group, which may appear within an **onInitiator**, **onTarget**, or **onSystem** element within an abstraction definition (*abstractionDefinition/ports/port/transactional/onmode*).



5.10.2 Description

A **transactionalPort** group contains elements defining constraints associated with a transactional logical port within an **abstractionDefinition**. It contains the following elements:

- a) **presence** (optional; default: **optional**) provides the capability to require or forbid a port to appear in a **busInterface**. Its three possible values are **illegal**, **required**, or **optional**. If this element is not present, its effective value is **optional**.
- b) **initiative** (optional; type: *string*; default: **requires**) defines the type of access: **requires**, **provides**, or **both**. For example, a SystemC *sc_port* is defined using **requires**, since it requires a SystemC interface.
- c) **kind** (optional) specifies the transactional port's type. It can take one of the following *enum* values:

tlm_port, **tlm_socket**, **simple_socket**, **multi_socket**, or **custom**. When **custom**, a *name* can be specified in the **custom** attribute. Also, the **tlm_port** should be used only for TLM1 notation; the other **enum** values should be used for TLM2 sockets.

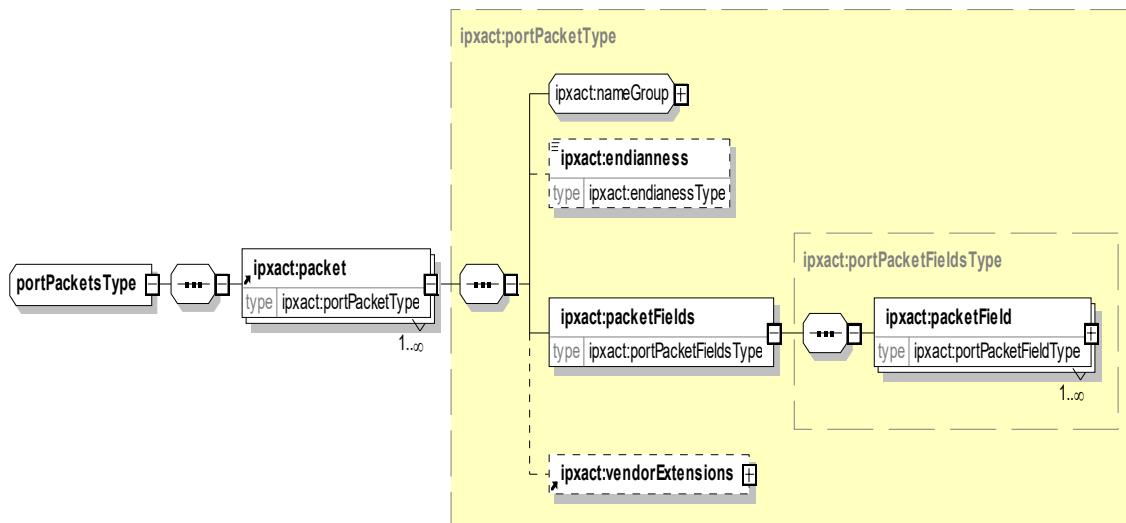
- d) **busWidth** (optional; type *unsignedPositiveIntExpression*, see [C.7.13](#)) specifies the data width in bits, e.g., 32 or 64.
- e) **protocol** (optional) defines how the information is transported by the transactional port (see [6.15.20](#)).

See also [SCR 6.2](#), [SCR 6.3](#), [SCR 6.4](#), [SCR 6.8](#), and [SCR 6.16](#).

5.11 Packets

5.11.1 Schema

The following schema describes the **abstractionDefinition/ports/port/packets** element.



5.11.2 Description

A port may contain an optional **packets** element which contains one or more **packet** elements. Each **packet** contains:

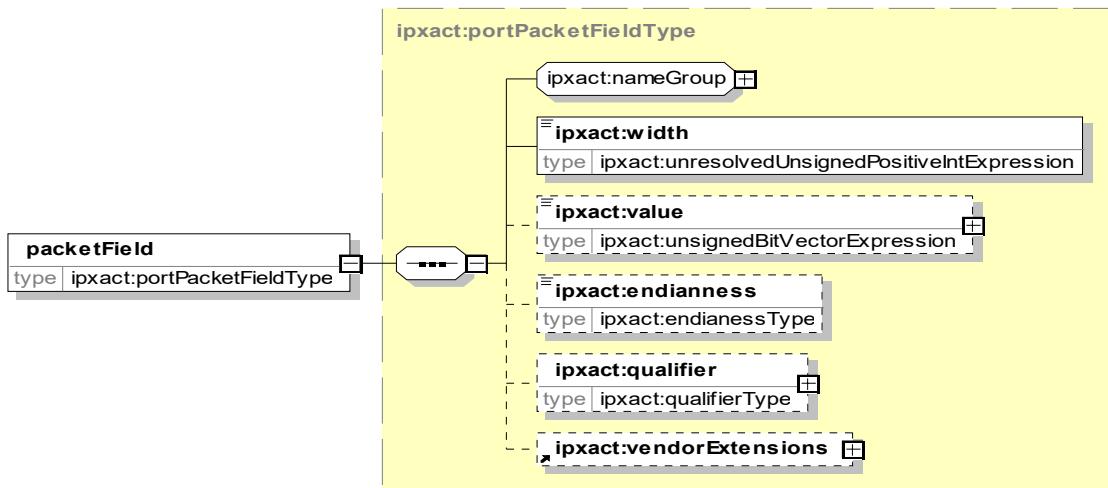
- a) The **nameGroup** group that contains a name, displayName, shortDescription, and description.
- b) **endianess** (optional) element: If not specified, the value is inherited from the busDefinition associated with this abstractionDefinition. Endianness applies to the bit ordering in all fields. See also [C.12](#).
- c) One or more **packetField** elements. See [5.11.3](#) and [Annex E](#).
- d) **vendorExtensions** (optional) contains any extra vendor-specific data related to the port. See [C.27](#).

Also see [SCR 6.47](#), [SCR 6.48](#), and [SCR 6.49](#).

5.11.3 packetField

5.11.3.1 Schema

The following schema details the information contained in the **packetField** element.



5.11.3.2 Description

One or more **packetField** elements, each of which contains:

- A **nameGroup** group that contains a name, displayName, shortDescription, and description.
- width** (mandatory; type **unresolvedUnsignedPositiveIntExpression**) describes the width of this packet field in number of bits. The SystemVerilog expression can use \$ipxact_packetfield_value() to reference other **packetField** elements in this packet. It can use \$ipxact_absdefport_value() to reference other port elements in the abstractionDefinition. See [E.1](#).
- value** (optional; type **unsignedBitVectorExpression**) describes the value of this packet field. If this packet field has a qualifier **isOpCode**, then the value has to be a constant.
- endianess** (optional; type: **endianessType**) describes the endianess of the packet field. If not specified, the value is inherited from the enclosing packet.
- qualifier** (optional; type: **qualifierType**) describes qualifiers for this packet field. See [5.6](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the packet field. See [C.27](#).

Also see [SCR 6.49](#).

5.12 Extending bus and abstraction definitions

5.12.1 Extending bus definitions

Bus definitions may use the **extends** element to create a family of compatible interconnectable bus definitions. A bus definition (B) extends another existing bus definition (A) by specifying the **extends** element in the B bus definition's element list. Bus definition B is referred to as the *extending* bus definition, and bus definition A is referred to as the *extended* bus definition. For two bus definitions related by the **extends** relation to be interconnectable, they need to be in a direct line of descent in the hierarchical *extension tree*, as illustrated in [Figure 8](#).

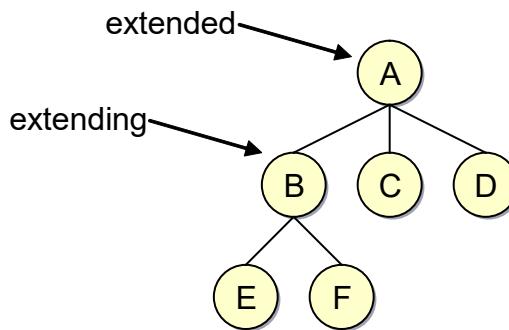


Figure 8—Extends relation hierarchy tree

In Figure 8, bus definition B extends bus definition A. Bus interfaces of bus definition E shall be connected only with bus interfaces of bus definitions E, B, and A. By the same token, bus interfaces of bus definition F shall be connected only with bus interfaces of bus definitions F, B, and A.

5.12.2 Extending abstraction definitions

The **abstractionDefinition** that references the *extended busDefinition* via the **busType** element is referred to as the *extended abstractionDefinition*. The bus definition writer shall supply an **abstractionDefinition** that references the *extending busDefinition*, and it is referred to as the *extending abstractionDefinition*. The *extending abstractionDefinition* shall reference the *extended abstractionDefinition* via its **extends** element. An example of extending is shown in Figure 9.²⁴

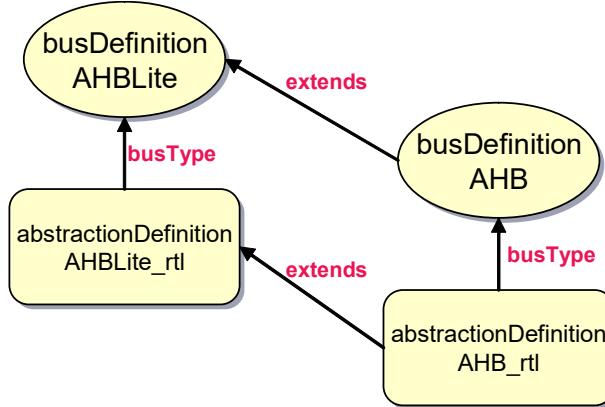


Figure 9—Example of extending

The *extending bus definition and abstraction definition pair* shall be able to stand on its own independent of the *extended bus definition and abstraction definition pair*; therefore, all the elements and attributes of the *extended bus definition and abstraction definition pair* shall be specified in the *extending bus definition and abstraction definition pair*. Also, all the ports in the *extended abstraction definition* shall be explicitly defined in the *extending abstraction definition*. Some of the elements and attributes of the *extending bus definition and abstraction definition pair* may be modified from the *extended bus definition and abstraction definition pair*, while others may not.

See also [SCR 3.16](#).

²⁴AHB = AMBA® high-speed bus. AMBA is an open specification on-chip backbone for interconnecting IP blocks. AMBA is a registered trademark of Arm Limited. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of this product. Equivalent products may be used if they can be shown to lead to the same results.

5.12.3 Modifying definitions

[Table 1](#) specifies which elements and attributes may be modified in a bus definition. The *Modifiable* column should be checked after elaboration.

Table 1—Elements of extending bus definition

Item	Modifiable	Note
directConnection	No	Covered by SCR 6.35
broadcast	No	Covered by SCR 6.36
isAddressable	No	Covered by SCR 6.37
maxInitiators	Yes	Smaller number applies when connecting interfaces of extended bus definitions. Covered by SCR 6.38
maxTargets	Yes	Smaller number applies when connecting interfaces of extended bus definitions. Covered by SCR 6.39
systemGroupNames	Yes	New group names may be added; group names not specified are not allowed by this bus definition. Covered by SCR 6.40
description	Yes	
parameters	Yes	
assertions	Yes	
vendorExtensions	Yes	

[Table 2](#) specifies which elements and attributes may be modified in an abstraction definition. The *Modifiable* column should be checked after elaboration.

Table 2—Elements of extending abstraction definition

Item	Modifiable	Note
ports	Yes	See Table 3 and SCR 6.11 .
description	Yes	
parameters	Yes	
assertions	Yes	
vendorExtensions	Yes	

The *extending* abstraction definition may add new ports, and the *extending* abstraction definition may mark certain ports as illegal to disallow their use. [Table 3](#) specifies which port (qualifier) elements may be modified when extending bus definitions. The *Modifiable* column should be checked after elaboration.

Table 3—Elements of a port in an *extending* abstraction definition

Item	Modifiable	Note
logicalName	No	Changing this name implies a port that is different from the one in the <i>extended abstractionDefinition</i> .
requiresDriver	Yes	
isAddress	No	Covered by SCR 6.41
isData	No	Covered by SCR 6.41
isClock	No	Covered by SCR 6.41
isReset	No	Covered by SCR 6.41
isValid	No	Covered by SCR 6.41
isInterrupt	No	Covered by SCR 6.41
isClockEn	No	Covered by SCR 6.41
isPowerEn	No	Covered by SCR 6.41
isOpcode	No	Covered by SCR 6.41
isProtection	No	Covered by SCR 6.41
isFlowControl	No	Covered by SCR 6.41
isUser	No	Covered by SCR 6.41
isRequest	No	Covered by SCR 6.41
isResponse	No	Covered by SCR 6.41
onSystem/group	Yes	
presence	Yes	
width	Yes	
direction	No	Covered by SCR 6.42
modeConstraints	Yes	
mirroredModeConstraints	Yes	
defaultValue	Yes	This default can be used to set a value for the extended abstraction definition logical port, if this port is not mapped or its presence is marked as illegal.
initiative	No	Covered by SCR 6.43
kind	No	Covered by SCR 6.44
busWidth	No	Covered by SCR 6.45
protocol	No	Covered by SCR 6.46
vendorExtensions	Yes	

5.13 Clock and reset handling

Abstraction definitions shall include all the logical ports that can participate in the protocol of the bus; bus interfaces also need to map to the component all the logical ports that participate in the protocol of that bus interface subject to its **presence** (see [5.7.2](#)).

This requirement applies to clock and reset ports as much as it does to other ports. If the protocol of a bus is dependent on a clock or reset port, the abstraction definition for that bus shall include that clock or reset port. Similarly if the bus protocol at a bus interface is dependent on a particular clock or reset port, the port map of that bus interface shall include that port. The clock or reset port, however, does not need to exist as a port of the component implementation, since it may be mapped to a phantom port of the component (see [6.15.22.2](#)). Also, since multiple bus ports may be mapped to a single component port (and component ports may also participate in ad hoc connections), the clock routing is not required to match or be defined by the bus infrastructure.

In some cases, a component may have clock or reset ports that are not associated with and do not participate in the protocol of any bus interface, but do provide a clock or reset to the internal logic of the component instead, e.g., a processor clock. In such cases, the clock port should be included in a special purpose clock or reset bus interface, with an appropriate special purpose bus type, or not be mapped into any interface and connected using ad hoc connections instead.

6. Component descriptions

6.1 Component

An IP-XACT *component* is used to describe the meta-data associated with any IP that can be instantiated in a design. Examples include IP such as cores (e.g., processors, co-processors, DSPs), peripherals (e.g., memories, DMA controllers, timers, UART), buses (e.g., simple buses, multi-layer buses, cross bars, network on chip), or any other IP block that can be instantiated in a design. An IP-XACT component can be of two kinds: static or configurable. A DE cannot change a *static component*. A *configurable* (or *parameterized*) *component* has configurable elements (such as parameters) that can be configured by the DE, and these elements may also configure the RTL or TLM model.

An IP-XACT component can be a hierarchical object or a leaf object. *Leaf components* do not contain other IP-XACT components, while *hierarchical components* contain other IP-XACT sub-components. This can be nested by having hierarchical components that contain hierarchical components, etc.—leading to the concept of *hierarchy depth*. The IP being described may have a completely different hierarchical arrangement in terms of its implementation in RTL or TLM to that of its IP-XACT description. So, a description of a large IP component may be made up of many levels of hierarchy, but its IP-XACT description need be only a leaf object as that completely describes the IP. On the other hand, some IP can be described only in terms of a hierarchical IP-XACT description, no matter what the arrangement of the implementation hierarchy.

6.1.1 Schema

The schema in [Figure 10](#) details the information contained in the **component** element, which is one of the top-level elements in the IP-XACT specification used to describe a component.

6.1.2 Description

Each element of a **component** is detailed in the rest of this subclause; the main sections of a **component** are as follows:

- a) **documentNameGroup** describes **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. See [C.11](#). The **versionedIdentifier** or VLNV consists of four subelements for a top-level IP-XACT element. See [C.28](#).
- b) **typeDefinitions** (optional) specifies the existence of type definitions in a component. See [6.2](#).
- c) **powerDomains** (optional) specifies the power domains of a component. See [6.3](#).
- d) **busInterfaces** (optional) specifies all the interfaces for this component. A **busInterface** is a grouping of ports related to a function, typically a bus, defined by a bus definition and abstraction definition. See [6.7](#).
- e) **indirectInterfaces** (optional) specifies access points and access information for any indirect memory maps. See [6.8](#).
- f) **channels** (optional) specifies the interconnection between interfaces inside of the component. See [6.9](#).
- g) **modes** (optional) documents the operating modes of a component. See [6.10](#).
- h) **addressSpaces** (optional) specifies the addressable area as seen from **busInterfaces** with an interface mode of initiator. See [6.11.1](#).
- i) **memoryMaps** (optional) specifies the addressable area as seen from **busInterfaces** with an interface mode of target or from cpus. See [6.12](#).
- j) **model** (optional) specifies all the different views, ports, and model configuration parameters of the component. See [6.15](#).

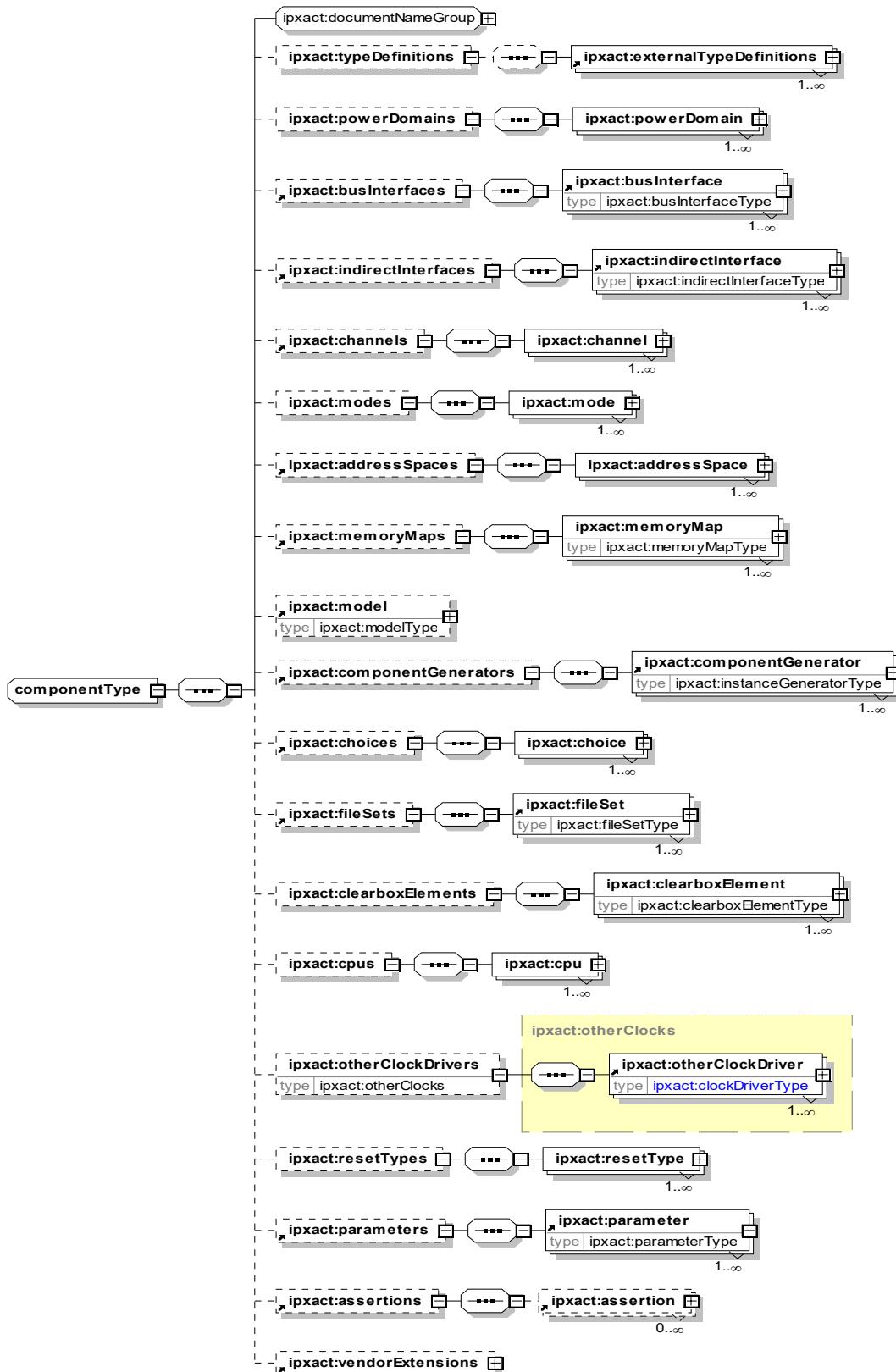


Figure 10—Component schema

- k) **componentGenerators** (optional) specifies a list of generator programs attached to this component. See [6.16](#).
- l) **choices** (optional) specifies multiple enumerated lists. These lists are referenced by other sections of this component description. See [C.8](#).
- m) **fileSets** (optional) specifies groups of files and possibly their function for reference by other sections of this component description. See [6.17](#).
- n) **clearboxElements** (optional) specifies all the different locations in the component that can be accessed for verification purposes. See [6.18](#).
- o) **cpus** (optional) indicates this component contains programmable processors. See [6.20](#).
- p) **otherClockDrivers** (optional) specifies any clock signals that are referenced by implementation constraints, but are not external ports of the component. See [6.15.18](#).
- q) **resetTypes** (optional) specifies user-defined reset types referenced within field reset elements. See [6.21](#).
- r) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this component. See [C.21](#).
- s) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.5](#).
- t) **vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See [C.27](#).

See also [SCR 1.10](#).

6.2 Type definitions

6.2.1 Schema

The following schema in [Figure 11](#) details the information contained in the **typeDefinitions** element, which may appear as an element inside the **component** element.

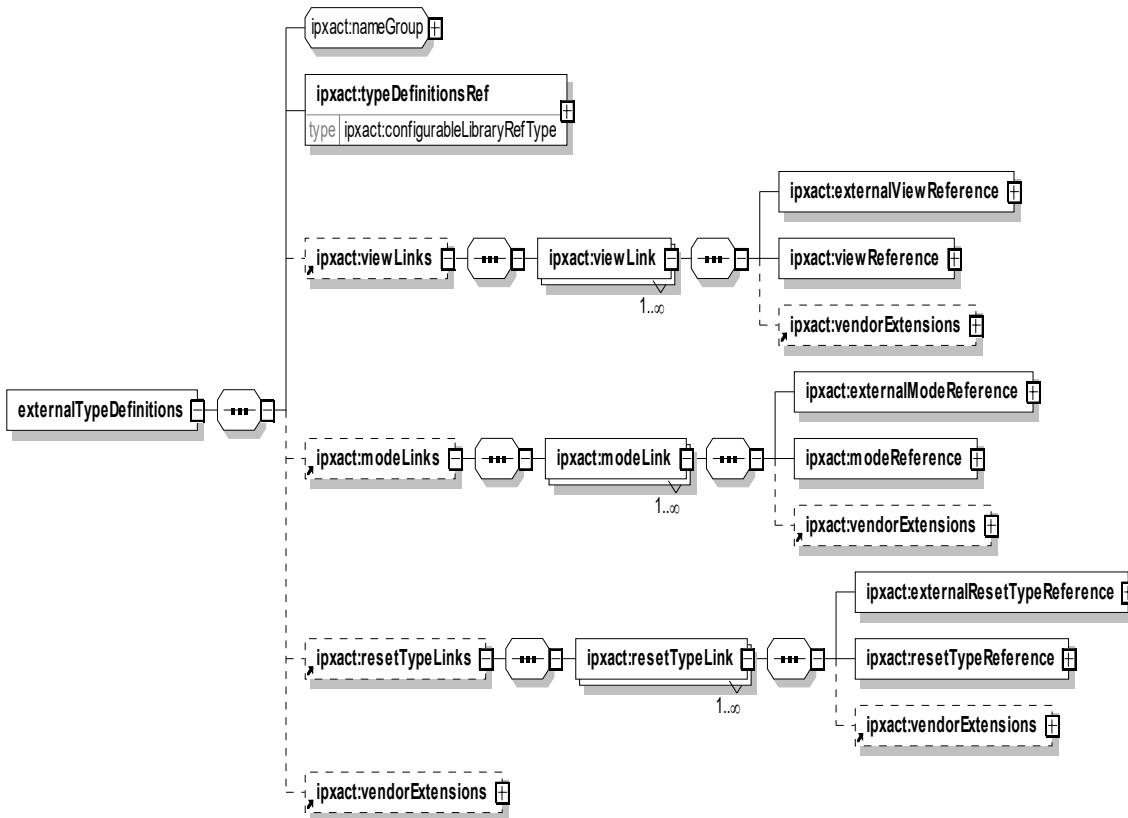


Figure 11—externalTypeDefinitions schema

6.2.2 Description

The **externalTypeDefinitions** instantiates and configures an external typeDefinitions document in its encapsulating component.

- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **typeDefinitions** element.
- b) **typeDefinitionsRef** (mandatory; type *configurableLibraryRefType*) is a reference to a typeDefinitions description for this externalTypeDefinitions. The **typeDefinitionsRef** element contains four attributes to specify a unique VLVN and an optional **configurableElementValues** element, which is used to document values for parameters to be passed into the referenced document. See also [C.10](#) and [9.1](#).
- c) **viewLinks** (optional) contains one or more **viewLink** elements. Each **viewLink** element binds a view of the referenced typeDefinitions to a view of the encapsulating top-level element (component or typeDefinitions).
 - 1) **externalViewReference** (mandatory) is a reference to a view of the referenced typeDefinitions.
 - 2) **viewReference** (mandatory) is a reference to a view of the encapsulating top-level element (component or typeDefinitions).
 - 3) **vendorExtensions** (optional) add any extra vendor-specific data related to the **viewLinks**. See [C.27](#).

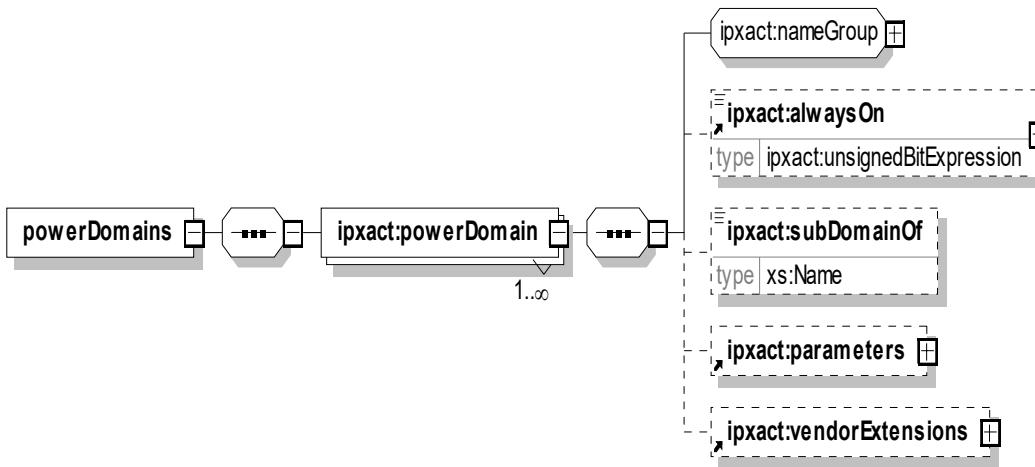
- d) **modeLinks** (optional) contains one or more modeLink elements. Each modeLink element binds a mode of the referenced typeDefinitions to a mode of the encapsulating top-level element (component or typeDefinitions).
 - 1) **externalModeReference** (mandatory) is a reference to a mode of the referenced typeDefinitions.
 - 2) **modeReference** (mandatory) is a reference to a mode of the encapsulating top-level element (component or typeDefinitions).
 - 3) **vendorExtensions** (optional) add any extra vendor-specific data related to the modelink. See [C.27](#).
- e) **resetTypeLinks** (optional) contains one or more resetTypeLink elements. Each resetTypeLink element binds a resetType of the referenced typeDefinitions to a resetType of the encapsulating top-level element (component or typeDefinitions). The resetType **HARD** is linked implicitly and shall not be referenced explicitly. See [6.21.2](#).
 - 1) **externalResetTypeReference** (mandatory) is a reference to a resetType of the referenced typeDefinitions.
 - 2) **resetTypeReference** (mandatory) is a reference to a resetType of the encapsulating top-level element (component or typeDefinitions).
 - 3) **vendorExtensions** (optional) add any extra vendor-specific data related to the resetTypeLink. See [C.27](#).
- f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the type definition. See [C.27](#).

See also [SCR 1.19](#), [SCR 1.20](#), [SCR 1.21](#), [SCR 1.22](#), [SCR 1.23](#), and [SCR 1.24](#).

6.3 Power domains

6.3.1 Schema

The following schema details the information contained in the **powerDomains** element, which may appear as an element inside the **component** element.



6.3.2 Description

The **powerDomain** documents a power domain of the encapsulating component.

- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **powerDomains** element.
- b) **alwaysOn** (optional; type *unsignedBitExpression*) describes if this power domain is always powered (if the value evaluates to 1) or not (if the value evaluates to 0).
- c) **subDomainOf** (optional; type *Name*) describes that this power domain is contained in another power domain of this component. The containing power domain is referenced.
- d) **parameters** (optional) describes any parameter that can be used to configure or hold information related to this component. See [C.21](#).
- e) **vendorExtensions** (optional) adds any extra vendor-specific data related to the power domain. See [C.27](#).

Component power domains can be linked to component instance power domains. See [7.2.2](#).

6.4 Interfaces

Each IP component normally identifies one or more bus interfaces. *Bus interfaces* are groups of ports that belong to an identified bus type (i.e., a reference to a **busDefinition** (see [5.2](#)) and abstraction types (i.e., references to **abstractionDefinitions** (see [5.3](#))). The purpose of the bus interface is to map the physical ports of the component to the logical ports of the abstraction definitions. This mapping provides more information about the interface.

There are seven possible modes for a bus interface. A bus interface may be an initiator, target, or system interface and may be direct or mirrored. The seventh interface mode is the monitor mode. A monitor interface can be used to connect IP into the design for verification.

6.4.1 Direct interface modes

An initiator *interface* is the interface mode that initiates a transaction (like a read or write) on a bus. Initiator interfaces tend to have *associated address spaces*.

A target *interface* is the interface mode that terminates or consumes a transaction initiated by an initiator interface. Target interfaces often contain information about the registers that are accessible through the target interface.

A system *interface* is neither an initiator nor target interface; this interface mode allows specialized (or non-standard) connections to a bus, such as external arbiters. System interfaces can be used to handle situations not covered by the bus specification or deviations from the bus specification standard.

The following guidelines also apply to the direct interface modes:

- If a port participates in the protocol of the initiator or target interfaces, it shall be included in initiator and target interfaces. System interfaces often contain some of the same ports as initiator or target interfaces.
- Some buses have specialized sideband ports. If these are tied or related to the standard ports in the bus (as opposed to being completely stand-alone), these ports should have a **system** element designator in the bus definition.

6.4.2 Mirrored interface modes

As the name suggests, a *mirrored interface* has the same (or similar) ports to its related direct bus interface, but each port's direction or initiative is reversed. So a port that is an input on a direct bus interface would be

an output in the matching mirrored interface. A mirrored bus interface (like its direct counterpart) supports the initiator, target, and system interface modes.

6.4.3 Monitor interface modes

A *monitor interface* connects to an initiator, target, system, mirrored-initiator, mirrored-target, or mirrored-system for observation. The connection shall not modify the connected interfaces. A monitor interface is identified by using the **monitor** element in the interface definition and specifying the type of active interface being monitored (e.g., initiator, target).

6.5 Interface interconnections

IP-XACT provides for three different types of connections between interfaces. A *direct connection* is a connection between an initiator interface and a target interface. A *mirrored-non-mirrored connection* is a connection between a direct interface and its corresponding mirrored interface (i.e., target and mirrored-target). A *monitor connection* is a connection between any interface type (other than monitor) and a monitor interface. It is not possible to connect two mirrored interfaces.

All interconnections are described in a top-level design object. See [7.1](#).

6.5.1 Direct connection

A direct connection is a connection between an initiator interface and a target interface. This connection is typically a point-to-point connection, but may be broadcast (one-to-many) under certain conditions (see [6.5.4](#)). More complex connection schemes with direct connections are also possible with the use of a component containing a **bridge** element(s).

See also [SCR 2.2](#), [SCR 2.11](#), [SCR 2.12](#), [SCR 2.13](#), and [SCR 2.14](#).

6.5.2 Mirrored-non-mirrored connection

A mirrored-non-mirrored connection is a connection between an initiator interface and a mirrored-initiator interface, a target interface and a mirrored-target interface, or a system interface and a mirrored-system interface. These connections are typically point-to-point, but may be broadcast (one-to-many) under certain conditions (see [6.5.4](#)). More complex connection schemes with mirrored-non-mirrored connections are also possible with the use of a component containing a **channel** element.

See also [SCR 2.2](#), [SCR 2.13](#), and [SCR 2.14](#).

6.5.3 Monitor connection

A monitor connection is a connection between a monitor interface and any other interface mode: initiator, mirrored-initiator, target, mirrored-target, system, or mirrored-system interface. The monitor interface is defined for only one mode and can be used only with that specific mode. Monitor connections are purely for non-intrusive observation of an interface. These connections are single-point-to-multi-point connections: the single point being the interface to be monitored and the multi-point being the monitor interface. More than one monitor may be attached to the same interface. The monitor connection shall meet the following:

- a) The connection of a monitor interface shall not count as a connected interface in the determination of the maximum initiator or maximum target calculations.
- b) The direction or initiative of ports in a monitor interface cannot be specified in an abstraction definition. All wire ports on a monitor interface shall be treated as having a logical direction of **in**. A monitor interface connected to any active interface shall see the values on the wire ports of the

active interface as inputs on its ports regardless of the direction they have on the active interface. All transactional ports on a monitor interface shall be treated as having a logical initiative of **provides**.

See also [SCR 2.2](#), [SCR 4.6](#), and the SCRs in [Table B.4](#).

6.5.4 Broadcast connections

IP-XACT interface connections are typically point-to-point, but broadcast (one-to-many) interface connections are allowed when the following conditions are met:

- a) The **busDefinition** associated with the connected interfaces has the **broadcast** element set to **true**.
- b) For each referenced logical port, the connections implied by connecting the associated physical ports shall not result in the creation of a signal with multiple drivers.

See also [SCR 2.17](#).

6.5.5 Interface logical to physical port mapping

An interface on a component can have multiple port-maps per abstraction-type to associate the physical ports on the component with the logical ports in the abstraction definition. This mapping is what provides the extra information needed to enable a higher level of design.

A *physical port* references a port defined in the component. Optionally **subPorts** and a slice of the port or **subPorts** can be referenced using partSelect elements. Packed bits of the port (slice) can be linearized according to the SystemVerilog IEEE Std 1800 subclause 7.4.

A *physical port* may have multiple vectors representing multiple dimensions. A *physical port* may also have multiple arrays. Each array element has a left and right element to represent the array in the same as in a vector. Vectors represent packed dimensions and arrays represent unpacked dimensions.

A *logical port* defined in an abstraction definition is assigned a logical port name and, optionally, a width. The logical port is assigned a numbering from `width-1` down to `0` if the width is present. If the width is not present, the logical port number shall have a lower bound of `0` and does not have an upper bound.

6.5.5.1 Mapping rules

Mapping rules describe the assignment of logical bit numbers to physical bit numbers.

- a) First, apply all the rules defined in [B.1.7](#) to determine the logical and physical ranges.
- b) The mapping is `logical.left → physical.left` down to `logical.right → physical.right`.

6.5.5.2 Physical interconnections

With all logical bits having been assigned from the abstraction definition to physical port, it is a simple matter to describe the physical connections that result from an interface connection. All connections are made purely based on the logical bit assignment. Like logical bit numbers from each interface are connected. The alignment is always such that logical bit `0` from interface A connects to logical bit `0` from interface B, logical bit `1` from interface A connects to logical bit `1` from interface B, and so on.

6.6 Complex interface interconnections

There are two constructs used to connect interfaces of standard components together (traditional components, usually with initiator and target interfaces): a channel and a bridge. These constructs are

encapsulated into components. Not only does the **channel** or **bridge** component provide a connection between standard components, but it also provides information on the addressing and data flow. With this information, it is possible to construct things such as a memory map for the system.

A *channel* identifies interfaces in a component that connect a component's initiator, target, and system interfaces on the same bus. All initiators connected to a channel see each target at the same physical address. On a channel, only one initiator may initiate transactions at a time. This does not preclude bus protocols that utilize pipelining or out-of-order completion. A bus that has addresses that are simultaneously seen differently from different initiators or a bus that allows transactions from different initiators to be simultaneously initiated may be represented only using bus bridges, not channels.

A *bridge* is an interface between two separate buses, which may be of the same or different types. Such a component has at least one initiator interface (onto the peripheral bus) and one target interface (onto the main system bus). Crossbar bus infrastructure (e.g., an ARM Multilayer AMBA) is also treated as a component containing bus bridges—such examples might have multiple initiator and multiple target interfaces.

6.6.1 Channel

A *channel* is a general name that denotes the collection of connections between multiple internal bus interfaces. The memory map between these connections is restricted so that, for example, a generator can be called to automatically compute all the address maps for the complete design. A channel can represent a simple wiring interconnect or a more complex structure such as a bus.

A channel also encapsulates the connection between initiator and target components. A channel is the construct that represents the bus infrastructure and allows transactions initiated by an initiator interface to be completed by a target interface.

The following rules apply for using channels:

- a) A target connected to a channel has the same address as seen from all initiators connected to this channel. Therefore, the target addresses (as seen by each initiator) are consistent for the system. As a consequence, all target interfaces connected to a channel see the same address. (If they do not, they are connected to different channels.)
- b) A channel supports memory mapping and remapping (see [6.12](#), [6.13](#), [Clause 13](#), and [H.3](#)).

See also [SCR 3.2](#).

6.6.2 Bridge

Some buses can be modeled using a component as a bridge. A *bridge* is a component that physically links one or more initiator bus interfaces to a target bus interface and logically connects the initiator address space(s) to a target memory map having two bus types on each side. This component has at least one initiator bus interface and at least one target bus interface, each for different protocols, and the bridge translates any signals between them. The target bus's interface definition references a memory-map that contains **subspaceMaps** (references initiator-interfaces, see [6.12.9](#)), or the definition contains a **transparentBridge** element (or a set of them, see [6.7.6.2](#)) to designate the corresponding initiator bus interface(s). There are two different types of bridges defined in IP-XACT: transparent and opaque. See also [Annex H](#).

The bridge relationship is *transparent* (a **transparentBridge** element is used, see [6.7.6.2](#)) when the address space on the bridged initiator bus interface is a decoded subset of the main address space, as seen through the bus bridge's target bus interface. In this case, a target component connected on the bridged initiator side shall reserve an address block on the main memory map seen on the bridging target side. If nothing is attached to the bridged initiator bus interface, then no address block is reserved on the main memory map.

The bridge relationship is *opaque* (the referenced memory-map contains **subspaceMaps**, see [6.12.9](#)) when the address space on the bridged initiator bus interface is not directly accessible to the main address space, as seen from the channel to which the target bus interface is connected. In this case, the bridging component occupies a single address block, which is the size of its target bus interface, reserved on the memory map of the initiators attached to the main bus channel.

The following rules apply for using bridges:

- a) A target interface can bridge to multiple address spaces. Specifically, a bridge shall have one or more initiator interfaces, and each initiator interface may have an address space associated with that interface.
- b) A bridge can have only direct interfaces. As a consequence, a bridge can directly connect to another component (e.g., initiator interface to target interface connection) under the conditions defined in [6.5.1](#). Or it can connect to a channel (e.g., initiator interface to mirrored-initiator interface).
- c) A bridge supports memory mapping and remapping (see [6.12](#), [6.13](#), [Clause 13](#), and [H.3](#)).

The transfer of addressing information from the target interface to the initiator interface of a bridge is done through the address space assigned to the initiator interface. This address space defines the visible address range from this initiator interface.

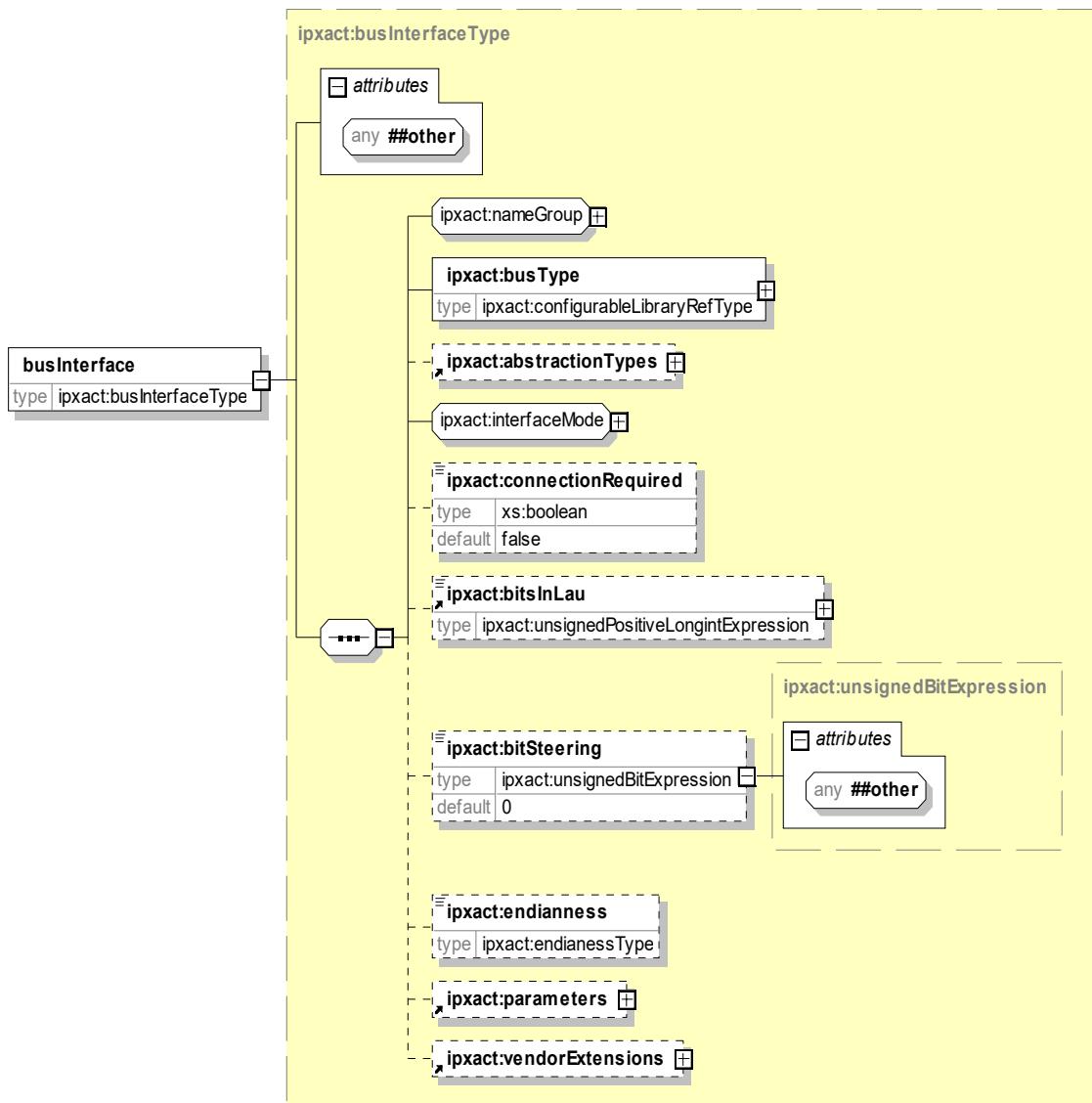
See also [SCR 9.1](#), [SCR 9.6](#), and [SCR 9.7](#).

6.7 Bus interfaces

6.7.1 busInterface

6.7.1.1 Schema

The following schema details the information contained in the **busInterfaces** element, which may appear as an element inside the top-level **component** element.



6.7.1.2 Description

Bus interfaces enable individual ports that appear on the component to be grouped together into a meaningful, known protocol. When the protocol is known, a lot of additional information can be written down about the characteristics of that interface.

The **busInterfaces** element contains an unbounded list of **busInterface** elements; therefore, a **component** may have multiple bus interfaces of the same or different types. Each **busInterface** element defines

properties of this specific interface in a component. The **busInterface** element also allows for vendor attributes to be applied. It contains the following elements and attributes:

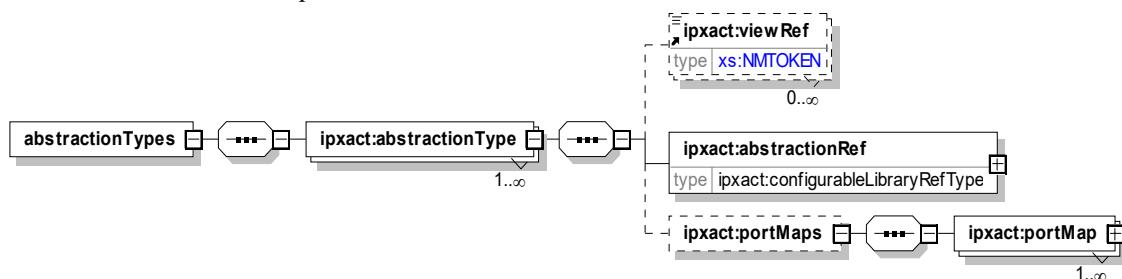
- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **component** element.
- b) **busType** (mandatory; type: *configurableLibraryRefType*, see [C.10](#)) specifies the bus definition that this bus interface is referenced. A bus definition (see [5.2](#)) describes the high-level attributes of a bus description.
- c) **abstractionTypes** (optional) specifies the abstraction definitions of the containing **busInterface**. An abstraction definition describes the low-level attributes of a bus (see [5.3](#)). The **abstractionTypes** element defines a list of **abstractionType** elements. Each abstraction type defines a particular abstraction being used for one or more views and the port mapping associated with the use of that particular abstraction. See [6.7.2](#).
- d) **interfaceMode** group describes further information on the *mode* for this interface. There are seven possible modes for an interface: initiator, target, mirroredInitiator, mirroredTarget, system, mirroredSystem, and monitor. See [6.7.4](#).
- e) **connectionRequired** (optional; type: *boolean*; default: **false**), if **true**, specifies when this component is integrated; this interface shall be connected to another interface for the integration to be valid. If **false**, this interface may be left unconnected. If this element is not present, its effective value is **false**.
- f) **bitsInLau** (optional; type: *unsignedPositiveLongintExpression*, see [C.7.14](#), default: **8**) describes the number of data bits that are addressable by the least significant address bit in the bus interface. It is appropriate to specify this element only for interfaces that are addressable.
- g) **bitSteering** (optional; default: **0**) designates if this interface has the ability to dynamically align data on different byte channels on a data bus. This element shall be specified only for interfaces that are addressable. The **bitSteering** element is an unsigned bit expression. A value of **1** indicates this interface uses data steering logic, and a value of 0 indicates this interface does not use data steering logic.
- h) **endianness** (optional) indicates the endianness of the bus interface. The two choices are **big** for big-endian and **little** for little-endian. If this element is not present, its effective value is **little**. See also [C.12](#).
- i) **parameters** (optional) specifies any parameter data value(s) for this bus interface. See [C.21](#).
- j) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this bus interface. See [C.27](#).

See also [SCR 1.4](#), [SCR 2.14](#), [SCR 2.15](#), [SCR 5.24](#), [SCR 5.25](#), [SCR 9.3](#), [SCR 9.4](#), and [SCR 9.5](#).

6.7.2 Abstraction types

6.7.2.1 Schema

The following schema defines the information contained in the **abstractionTypes** element, which appears within a **busInterface** description.



6.7.2.2 Description

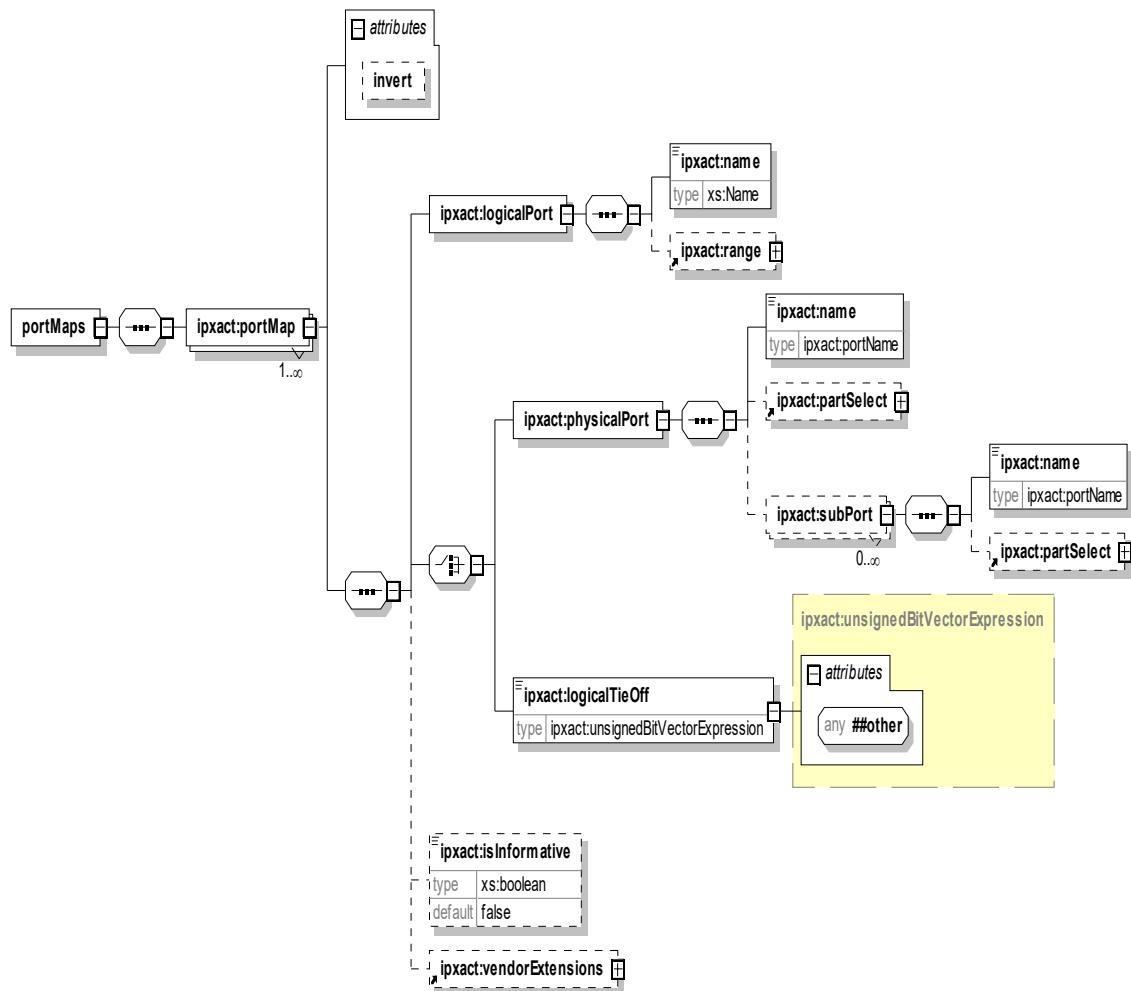
The **abstractionTypes** element specifies the abstraction definition where this bus interface is referenced. Each **abstractionType** contains the following elements:

- a) **viewRef** (optional; type: *NMTOKEN*) specifies the view to which the **abstractionType** applies. See [C.29](#).
 - b) **abstractionRef** (mandatory; type: *configurableLibraryRefType*, see [C.10](#)) is a reference to an abstraction definition description for this abstractionType instance.
 - c) **portMaps** (optional) describes the mapping between the abstraction definition's logical ports and the component's physical ports. See [6.7.3](#).

6.7.3 Port map

6.7.3.1 Schema

The following schema details the information contained in the **portMaps** element, which appears as an element inside the **abstractionType** element.



6.7.3.2 Description

Each **portMap** element describes the mapping between the logical ports, defined in the referenced abstraction definition, to the physical ports, defined in the containing component description.

- a) **invert** (optional; default: **false**) specifies connections to the indicated physical port shall be logically inverted.
- b) **logicalPort** (mandatory) contains the information on the logical port from the abstraction definition.
 - 1) **name** (mandatory; type: *Name*) specifies the logical port name. The name shall be a name of a logical port in the referenced abstraction definition that is defined as legal for this interface mode.
 - 2) **range** (optional) is used for a vectored logical port to specify the indices of the logical port mapping. See [C.25](#).
- c) **portMap** also contains one of the following:
 - 1) **physicalPort** (type: *Name*) contains information on the physical port contained in the component. See [B.1.7](#).
 - i) **name** (mandatory) specifies the physical port name. The name shall be a name of a port in the containing component.
 - ii) **partSelect** (optional) is used for a physical port (vectored, arrayed, or both) to specify the indices of the physical port mapping. If no **partSelect** is specified, the entire port is assumed to be connected. See [C.22](#).
 - iii) **subPort** (optional zero or more times). Sequence of subPort elements indicates an hierarchical path in a structured port. All subPort elements below this hierarchical path are assumed to be connected.
 - 4) **logicalTieOff** (type: *unsignedPositiveIntExpression*, see [C.7.13](#)) indicates the physical connection for this logical port is the specified constant value (instead of a physical port.)
- d) **isInformative** (optional; type *boolean*; default: **false**) when **true**, specifies this **portMap** is used only for information purposes. No connections will be made to the referenced physical port.
- e) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this bus interface. See [C.27](#).

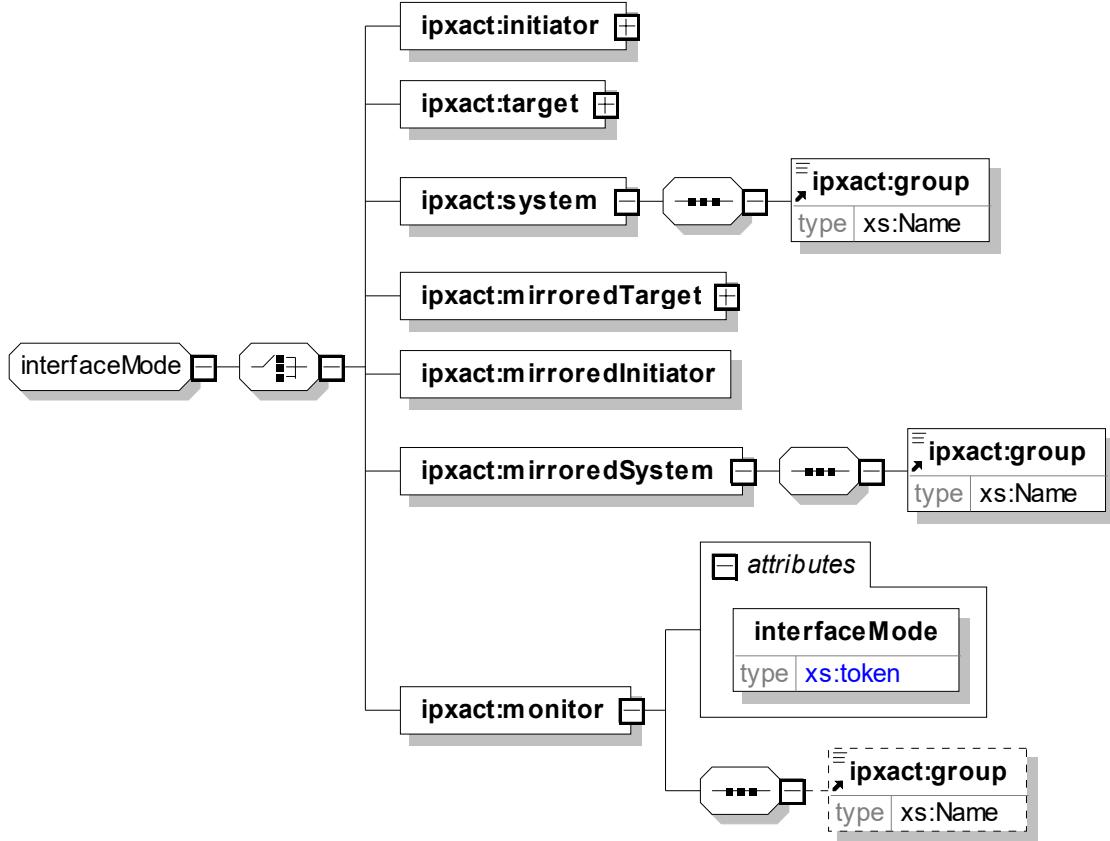
The same physical port may be mapped to a number of different logical ports on the same or different bus interfaces, and the same logical port may be mapped to a number of different physical ports. For port mapping rules, see [6.5.5.1](#).

See also [SCR 6.1](#), [SCR 6.2](#), [SCR 6.3](#), [SCR 6.4](#), [SCR 6.5](#), [SCR 6.6](#), [SCR 6.7](#), [SCR 6.12](#), [SCR 6.13](#), [SCR 6.17](#), [SCR 6.18](#), [SCR 6.19](#), [SCR 6.20](#), [SCR 6.21](#), [SCR 6.22](#).

6.7.4 Interface modes

6.7.4.1 Schema

The following schema details the information contained in the *interfaceMode* group, which appears as a group inside the **busInterface** element.



6.7.4.2 Description

The **busInterface** mode designates the purpose of the **busInterface** on this component. There are seven possible modes: three pairs of standard functional interfaces and their mirrored counterparts, and a monitor interface.

The *interfaceMode* group shall contain one of the following elements:

- An **initiator** interface mode is one that initiates transactions. See [6.7.5](#).
- A **target** interface mode is one that responds to transactions. See [6.7.6](#).
- A **system** interface mode is used for some classes of interfaces that are standard on different bus types, but do not fit into the initiator or target category.

The **group** (mandatory; type: *Name*) attribute for the **system** element defines the name of the group to which this system interface belongs.

- A **mirroredTarget** interface mode is the mirrored version of a target interface and can provide addition address offsets to the connected target interface. See [6.7.7](#).
- A **mirroredInitiator** interface mode is the mirrored version of an initiator interface.
- A **mirroredSystem** interface mode is the mirrored version of a system interface.

The **group** (mandatory; type: *Name*) attribute for the **mirroredSystem** element defines the name of the group to which this **mirroredSystem** interface belongs.

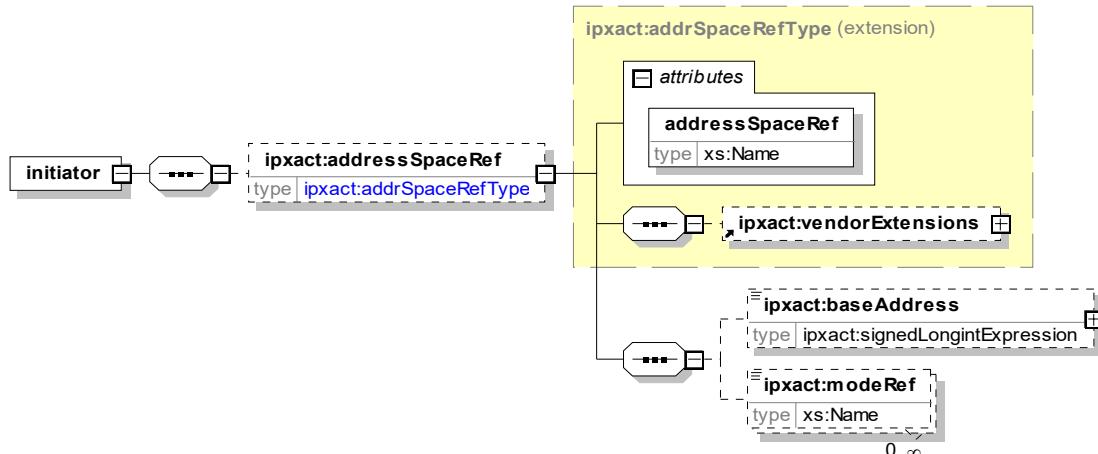
- g) A **monitor** interface mode is a special interface that can be used for verification. This monitor interface mode is used to gather data from other interfaces. See [6.5.3](#).
 - 1) The **interfaceMode** (mandatory type: *token*) attribute defines the interface mode to which this monitor interface can be connected: **initiator**, **target**, **system**, **mirroredInitiator**, **mirrored-Target**, or **mirroredSystem**.
 - 2) The **group** (optional type: *Name*) element is required if the **interfaceMode** attribute is set to **system** or **mirroredSystem**. This element defines the name of the system group for this monitor interface.

See also [SCR 2.13](#), [SCR 4.3](#), [SCR 4.4](#), and [SCR 6.15](#).

6.7.5 Initiator interface

6.7.5.1 Schema

The following schema details the information contained in the **initiator** element, which appears as an element inside the **interfaceMode** group inside **busInterface** element.



6.7.5.2 Description

An **initiator** interface is one that initiates transactions. The **initiator** element contains the following elements and attributes. **addressSpaceRef** (optional) element contains attributes and subelements to describe information about the range of addresses with which this initiator interface can generate transactions.

- a) **addressSpaceRef** (mandatory; type: *Name*) attribute references a name of an address space defined in the containing description. The address space shall define the range and width for transaction on this interface. See [6.11.1.2](#).
- b) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this bus interface. See [C.27](#).
- c) **baseAddress** (optional; type: *signedLongintExpression*, see [C.7.4](#)) specifies the starting address of the address space. The address space numbering normally starts at 0. Some address spaces may use *offset addressing* (starting at a number other than 0) so the base address element can be used to designate this information.

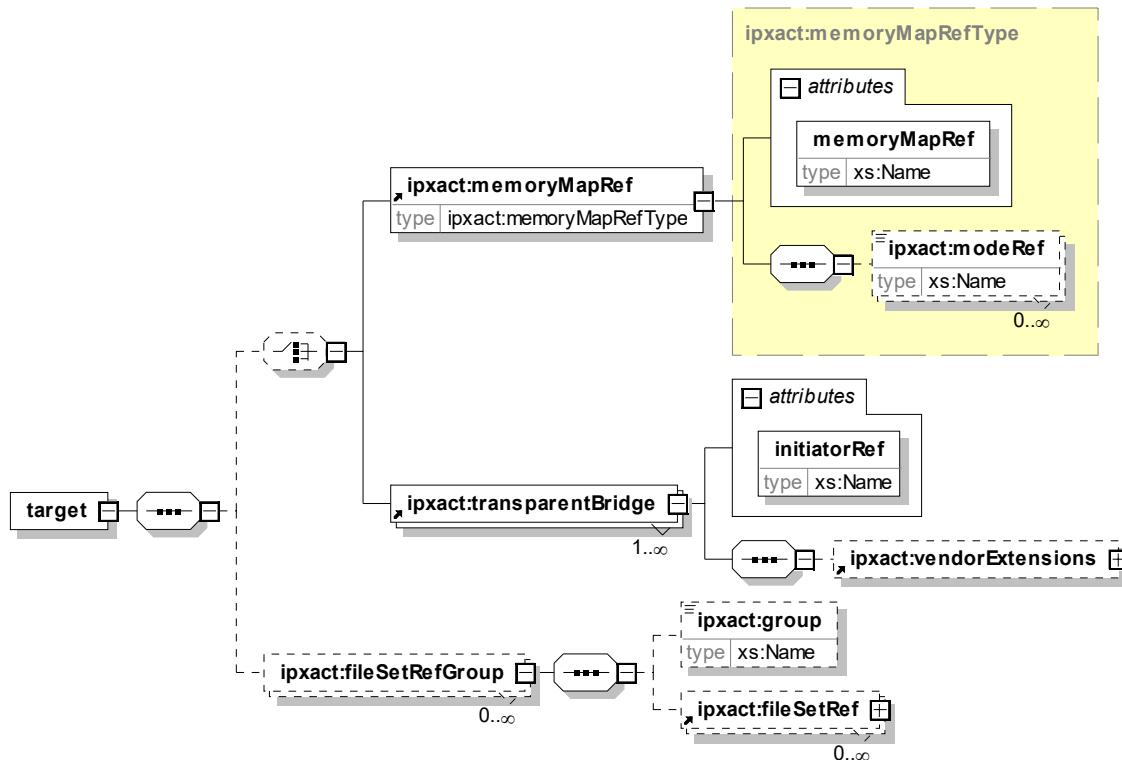
- d) **modeRef** (optional) references a mode of the component in which the referenced addressSpace is accessible. If no modeRef element is present, then access to the referenced addressSpace is not restricted by the initiator interface.

See also [SCR 9.1](#), [SCR 9.6](#), and [SCR 9.7](#).

6.7.6 Target interface

6.7.6.1 Schema

The following schema details the information contained in the **target** element, which appears as an element inside the **interfaceMode** group inside **busInterface** element.



6.7.6.2 Description

A **target** interface is one that responds to transactions. The memory map reference points to information about the range of registers, memory, or other address blocks accessible through this target interface. This target interface can also be used in a bridge application to “bridge” a transaction from a target interface to an initiator interface.

- a) A **target** can contain one of the following subelements:
- 1) **memoryMapRef** contains an attribute that references a memory map.
 - i) The **memoryMapRef** (mandatory; type: *Name*) attribute references a name of a memory map defined in the containing description. The memory map contains information about the range of registers, memory, or other address blocks. See [6.12](#).
 - ii) **modeRef** (optional) references a mode of the component in which the referenced memory map is accessible. If no modeRef element is present, then access to the referenced memory map is not restricted.

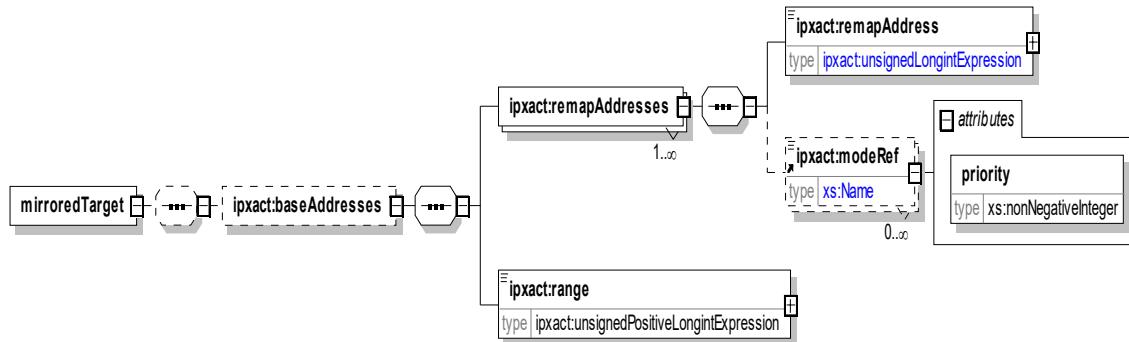
- 3) **transparentBridge** is an unbounded list of references to initiator interfaces. If the interface is of a bus definition that is addressable, a **transparentBridge** element may be included. A *transparent bridge* is one in which all addressing entering the target interface exits the above referenced initiator interface without any modifications.
- The **initiatorRef** (mandatory; type: *Name*) attribute shall reference an initiator interface (see [6.7.5](#)) in the containing description.
 - vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this bus interface. See [C.27](#).
- b) **fileSetRefGroup** (optional) element is an unbounded list of the references to file sets contained in this component. These file set references are associated with this target interface. This element allows each target interface to reference a unique **fileSet** element (see [6.17](#)) and can further be used to reference a software driver, which can be made different for each target interface.
- group** (optional; type: *Name*) element allows the definition of a group name for the **fileSetRefGroup**.
 - fileSetRef** (optional) is an unbounded list of references to a **fileSet** within the containing document. See [C.15](#).

See also [SCR 3.6](#), and [SCR 9.2](#).

6.7.7 Mirrored target interface

6.7.7.1 Schema

The following schema details the information contained in the **mirroredTarget** element, which appears as an element inside the *interfaceMode* group inside **busInterface** element.



6.7.7.2 Description

A **mirroredTarget** interface is used to connect to a target interface. The **mirroredTarget** interface may contain additional address information in the **baseAddresses** (optional) element.

- remapAddresses** (mandatory) is an unbounded list containing a **remapAddress** (mandatory; type: *unsignedLongintExpression*, see [C.7.12](#)) element that specifies the address offset to apply to the connected target interface. The **remapAddress** is expressed as the number of addressable units based on the size of an addressable unit as defined inside the containing **busInterface/bitsInLau** element.

The **modeRef** (optional) element identifies the operating **mode** for which the **remapAddress** element applies by referencing a mode name in the containing description and providing a priority. A **remapAddress** applies if **modeRef** references an active mode and that has highest

priority among the active modes referenced in all **remapAddress** elements of the **baseAddresses**. The **modeRef** element value shall be unique within the containing **baseAddresses** element. The **modeRef** element has an attribute **priority** (mandatory; type *nonNegativeInteger*) which indicates the priority of the referenced mode for this **remapAddress**. The lower the value, the higher the priority. The priority value of a modeRef element is not necessarily unique in the scope of the referenced modes inside the **baseAddresses** element. Hence, multiple **remapAddress** elements can apply.

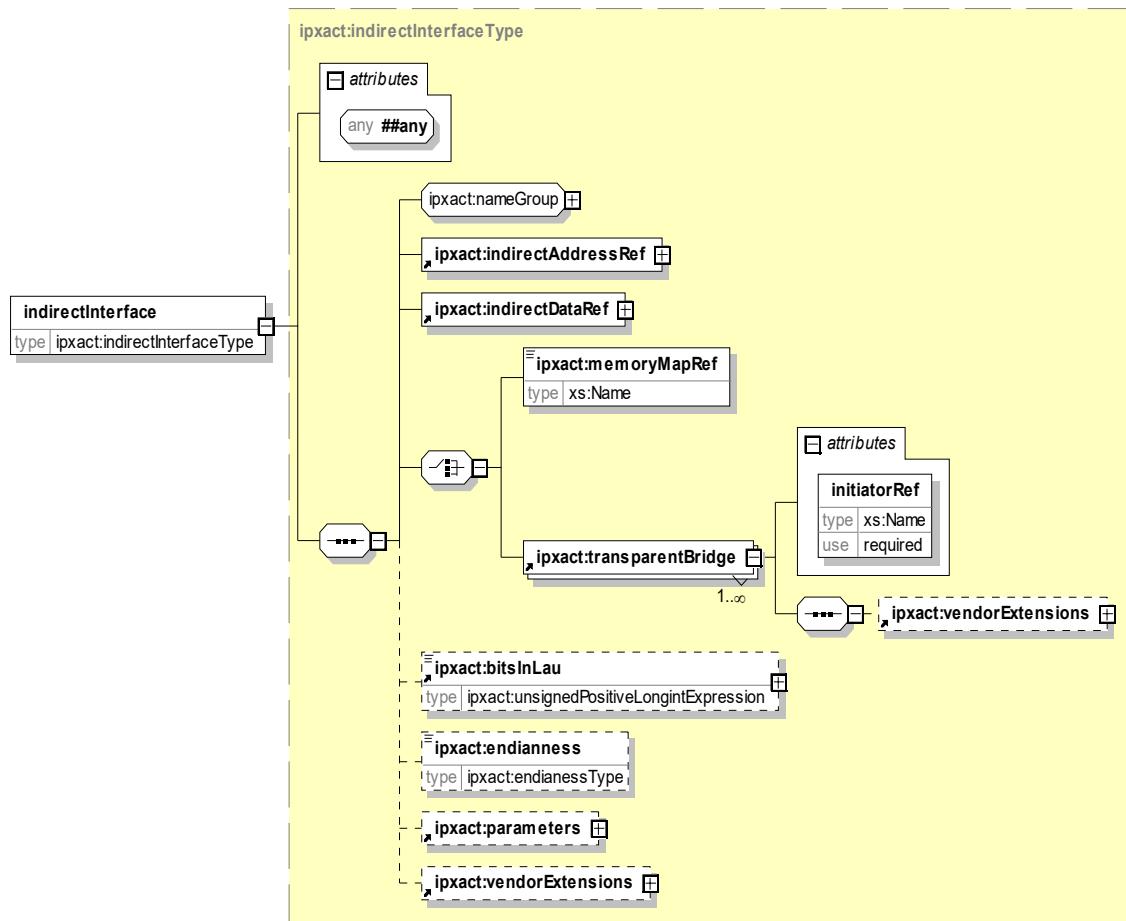
- b) **range** (mandatory; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) specifies the address range to apply to the connected target interface.

6.8 Indirect interfaces

The contents of an indirectly accessible memory map are not accessible at a fixed address via a bus interface. A bus interface accesses an indirectly accessible memory map via reads and writes to fields the bus interface can directly access.

6.8.1 Schema

The following schema details the information contained in the **indirectInterfaces** element, which may appear as an element inside the top-level **component** element.



6.8.2 Description

The **indirectInterfaces** element contains an unbounded list of **indirectInterface** elements. Each **indirectInterface** element defines the access details for an indirectly accessible memory map element. Any transaction on the referenced field of the **indirectDataRef** is redirected to the address contained in the **indirectAddressRef** of the indirectly accessible memory map.

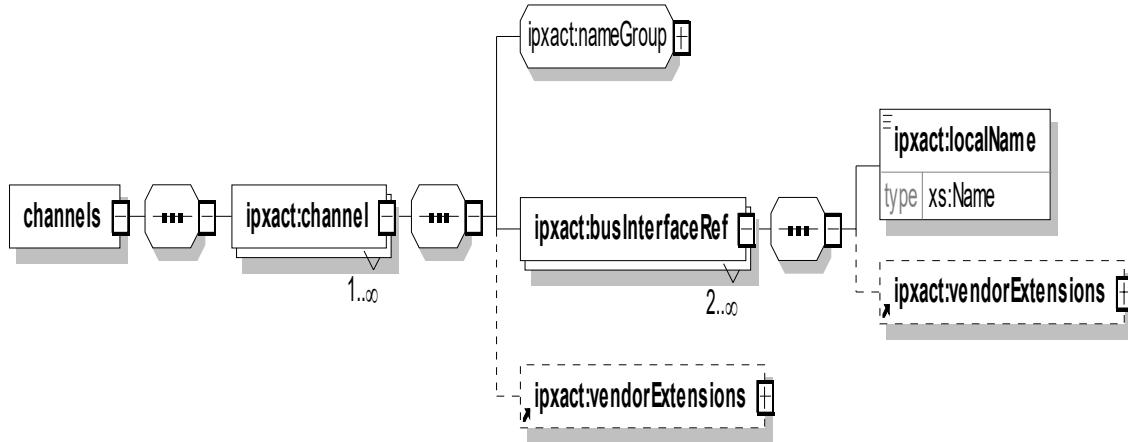
- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **indirectInterfaces** element.
- b) **indirectAddressRef** (mandatory) references the register field used for addressing the indirectly accessible memory map. The referenced field is usually directly accessible via a bus interface. This reference is made using a **fieldReferenceGroup** (see [C.13](#)).
- c) **indirectDataRef** (mandatory) references the register field used for read/write access for the indirectly accessible memory map. The field is usually accessible via a standard bus interface. This reference is made using a **fieldReferenceGroup** (see [C.13](#)).
- d) An **indirectInterface** element contains either a **memoryMapRef** element or one or more **transparentBridge** elements.
 - 1) **memoryMapRef** (type: *Name*) references a name of a memory map defined in the containing description. This memory map is indirectly accessible.
 - 2) **transparentBridge** is an unbounded list of references to initiator interfaces. If the interface is of a bus definition that is addressable, a **transparentBridge** element may be included. A *transparent bridge* is one in which all addressing entering the target interface exits the above referenced initiator interface without any modifications.
 - i) The **initiatorRef** (mandatory; type: *Name*) attribute shall reference an initiator interface (see [6.7.5](#)) in the containing description.
 - ii) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this bus interface. See [C.27](#).
- e) **bitsInLau** (optional; type: *unsignedPositiveLongintExpression* (see [C.7.14](#)); default: **8**) describes the number of data bits that are addressable by the least significant address bit in the bus interface.
- f) **endianness** (optional) indicates the endianness of the indirect interface. The two choices are **big** for big-endian and **little** for little-endian. If this element is not present, its effective value is **little**. See also [C.12](#).
- g) **parameters** (optional) specifies any parameter data value(s) for this indirect interface. See [C.21](#).
- h) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this indirect interface. See [C.27](#).

See also [SCR 9.8](#), [SCR 9.9](#), and [SCR 9.10](#).

6.9 Component channels

6.9.1 Schema

The following schema details the information contained in the **channels** element, which may appear as an element inside the top-level **component** element.



6.9.2 Description

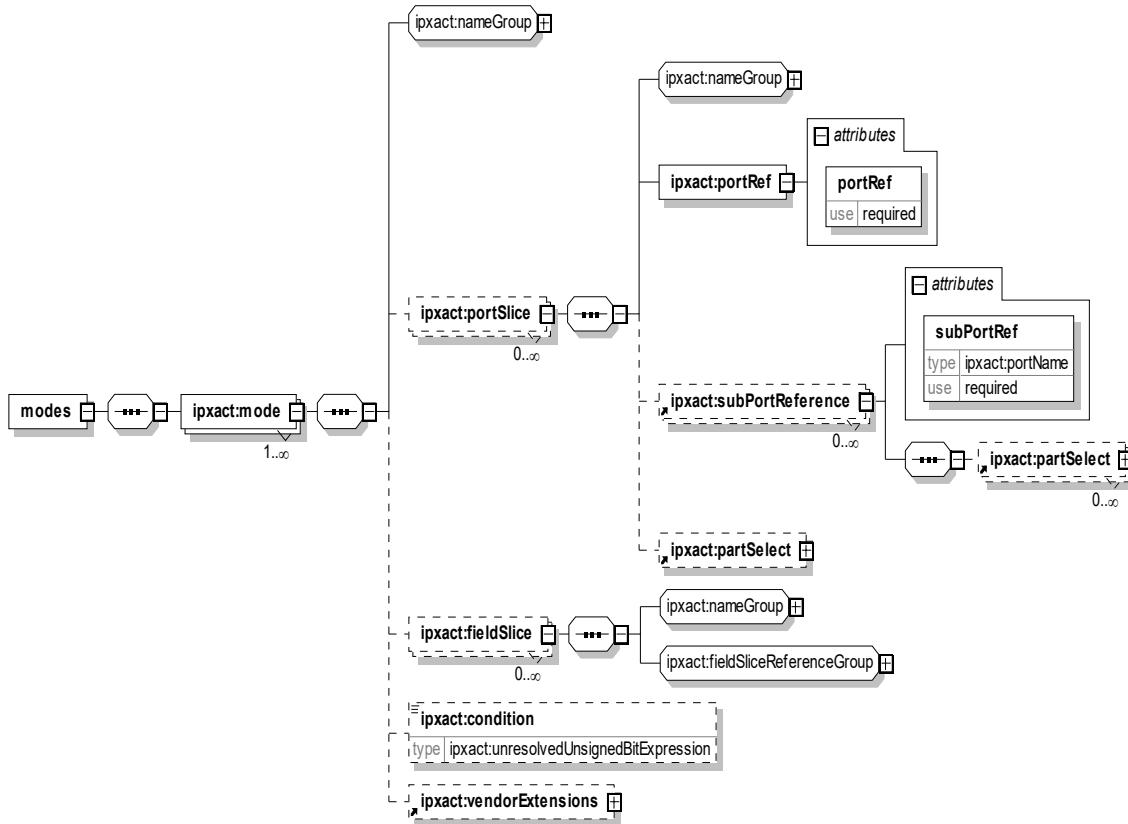
The **channels** element contains an unbounded list of **channel** elements. Each **channel** element contains a list of all the mirrored bus interfaces in the containing component that belong to the same channel.

- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **channels** element.
- b) **busInterfaceRef** (mandatory) is an unbound list of references (a minimum of two) to mirrored bus interfaces in the containing component. Each mirrored bus interface in a component may be referenced in any channel at most once. The order of this list may be used by the DE in some way and shall be maintained. See [6.7.1](#).
 - 1) **localName** (mandatory; type: *Name*) specifies the name of a mirrored bus interface within this channel.
 - 2) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this bus interface. See [C.27](#).
- c) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this channel. See [C.27](#).

6.10 Modes

6.10.1 Schema

The following schema details the information contained in the **modes** element, which may appear as an element inside the top-level **component** element.



6.10.2 Description

A **modes** element describes a set of one or more **mode** elements. Each mode element defines a conditional mode where each mode can be conditioned by a port slice specified with a **portSlice** element and/or a register field slice specified by a **fieldSlice** element.

- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **modes** element.
 - b) **portSlice** (optional) describes a slice of a component port. Subelements are **nameGroup**, **portRef**, **subPortReference**, and **partSelect**.
 - 1) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **modes** element.
 - 2) **portRef** (mandatory; type: **portName**) attribute is the name of the port in the containing description to which this logic value comparison is assigned. See [6.15.7](#).
 - 3) **subPortReference** references a subPort of a structured port that is referenced by **portRef**. Multiple **subPortReferences** form a path in the structured port
 - 4) **partSelect** (optional) defines the sub-range of the port being referenced. If no **partSelect** is specified, the entire port is assumed to be connected. See [C.22](#).

- c) **fieldSlice** (optional) describes a slice of a register field. Subelements are nameGroup and fieldSliceReferenceGroup. See [C.14](#).
- d) **condition** (optional; type: unresolvedUnsignedBitExpression) describes an expression expressed as an extension of the IP-XACT SystemVerilog expression language enabling port value references, register field value references, and other mode condition value references using the following syntax:

```
$ipxact_port_value( <portSliceName> )
$ipxact_field_value( <fieldSliceName> )
$ipxact_mode_condition( <modeName> )
```

It is assumed that values of component mode condition expressions are stable during transactions into the component memory maps. A mode is called active, if its condition evaluates to 1.

- e) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this mode. See [C.27](#).

See also [SCR 1.36](#), [SCR 1.37](#).

6.11 Address spaces

An address space is defined as a logical addressable space of memory. Each initiator interface can be assigned a logical address space. Address spaces are effectively the programmer's view looking out from an initiator interface. Some components may have one address space associated with more than one initiator interface (for instance, a processor that has a system bus and a fast memory bus). Other components (for instance, Harvard architecture processors) may have multiple address spaces associated with multiple initiator interfaces—one for instruction and the other for data.

6.11.1 addressSpaces

6.11.1.1 Schema

The following schema details the information contained in the **addressSpaces** element, which may appear as an element inside the top-level **component** element.



6.11.1.2 Description

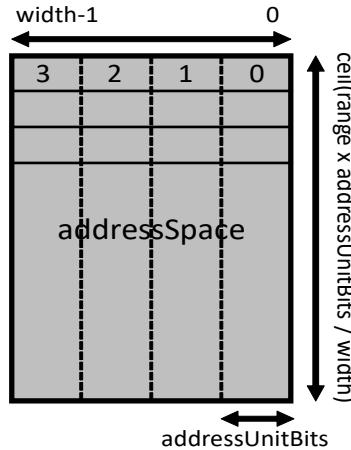
The **addressSpaces** element contains an unbounded list of **addressSpace** elements. Each **addressSpace** element defines a logical address space seen by an initiator bus interface. It contains the following elements:

- nameGroup** group is defined in [C.19](#).
- blockSize** group includes the following:
 - range** (mandatory; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) gives the address range of an address space. See [C.7.14](#).
 - width** (mandatory; type: *unsignedIntExpression*) is the bit width of a row in the address space. The type of this element is set to **nonNegativeInteger**. See [C.7.11](#).
- segments** (optional) describes a portion of the address space starting at an address offset and continuing for a given range. A **segment** can be referenced by a **subspaceMap**. See [6.11.2](#).
- addressUnitBits** (optional; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) defines the number of data bits in each address increment of the address space. If this element is not present, it is presumed to be 8.
- localMemoryMap** (optional) describes a local memory map that is seen exclusively by this initiator bus interface viewing this address space. See [6.11.3](#).

- f) **parameters** (optional) specifies any parameter data value(s) for this address space. See [C.21](#).
- g) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this address space. See [C.27](#).

The **range** and **width** elements are related by the following formulas:

$$\begin{aligned} \text{number_of_bits_in_block} &= \text{addressUnitBits} \times \text{range} \\ \text{number_of_rows_in_block} &= \text{ceil}(\text{number_of_bits_in_block} / \text{width}) \end{aligned}$$

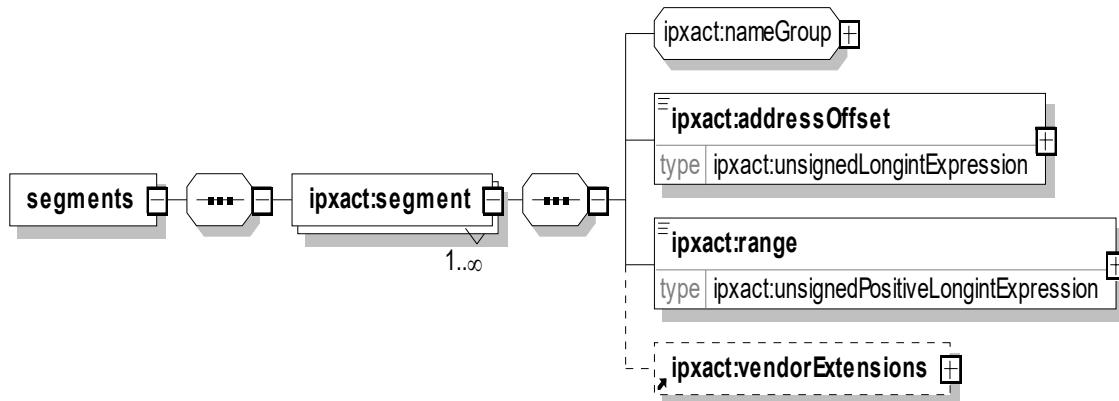


See also [SCR 9.1](#), [SCR 9.6](#), and [SCR 9.7](#).

6.11.2 Segments

6.11.2.1 Schema

The following schema details the information contained in the **segments** element, which may appear inside an **addressSpace** element.



6.11.2.2 Description

The **segments** element contains an unbounded list of **segment** elements. Each **segment** describes the location and size of an area in the containing **addressSpace**. The **segment** element contains the following elements:

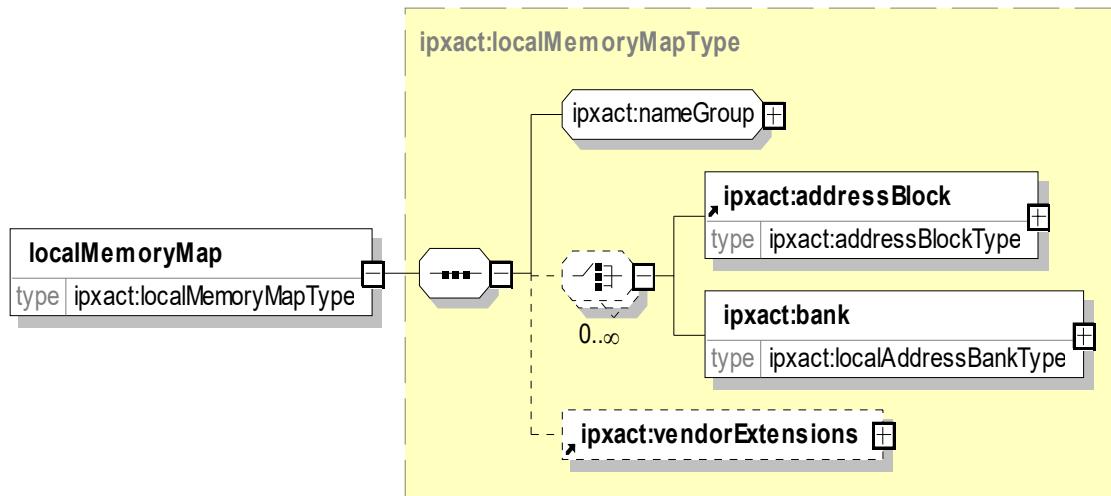
- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **segments** element.
- b) **addressOffset** (mandatory; type: *unsignedLongintExpression*) describes, in addressing units from the containing **addressSpace/addressUnitBits** element, the offset from the start of the **addressSpace**. See [C.7.12](#).
- c) **range** (mandatory; type: *unsignedPositiveLongintExpression*) gives the address range of an address space segment. This is expressed as the number of addressable units of the address space segment. The size of an addressable unit is defined inside the **addressUnitBits** element. See [C.7.14](#).
- d) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this address space. See [C.27](#).

See also [SCR 9.6](#).

6.11.3 Local memory map

6.11.3.1 Schema

The following schema details the information contained in the **localMemoryMap** element, which may appear inside an **addressSpace** element.



6.11.3.2 Description

Some processor components require specifying a memory map that is local to the component. *Local memory maps* (the **localMemoryMap** element in the **addressSpace** element of the component) are blocks of memory within a component that can be accessed only by the initiator interfaces of that component. If the encapsulating **addressSpace** is referenced by multiple initiator interfaces, then it is not specified if the **localMemoryMap** is shared. If the initiator interface containing a local memory map is bridged from a target

interface (see [6.6.2](#)), the local memory map is visible from this target interface. **localMemoryMap** contains the following elements:

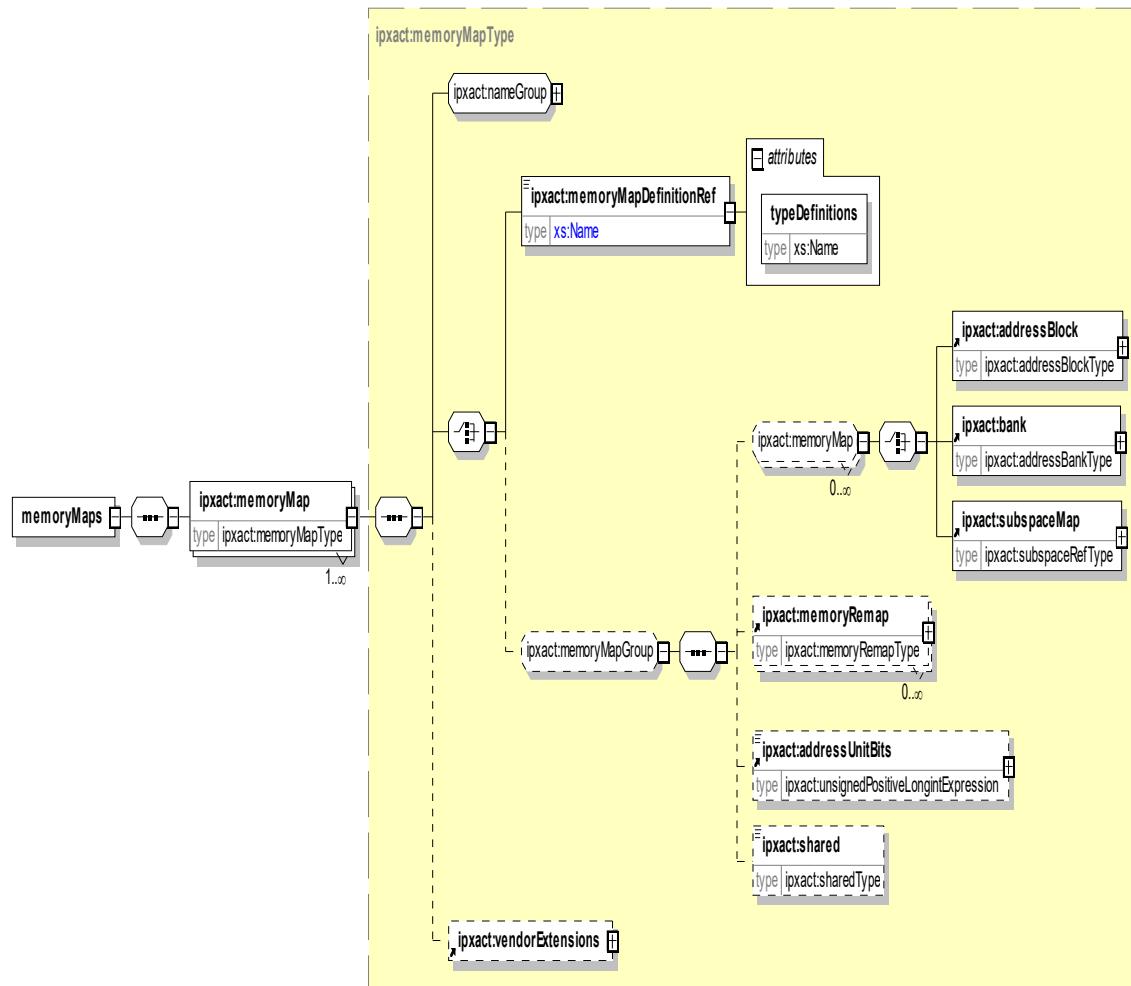
- a) **nameGroup** group is describe in [C.19](#).
- b) **localMemoryMap** is any number of the following:
 - 1) **addressBlock** describes a single block. See [6.12.2](#).
 - 2) **bank** represents a collection of address blocks, banks, or subspace maps. See [6.12.5](#).
- c) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this address space. See [C.27](#).

6.12 Memory maps

6.12.1 memoryMaps

6.12.1.1 Schema

The following schema details the information contained in the **memoryMaps** element, which may appear as an element inside the **component** element.



6.12.1.2 Description

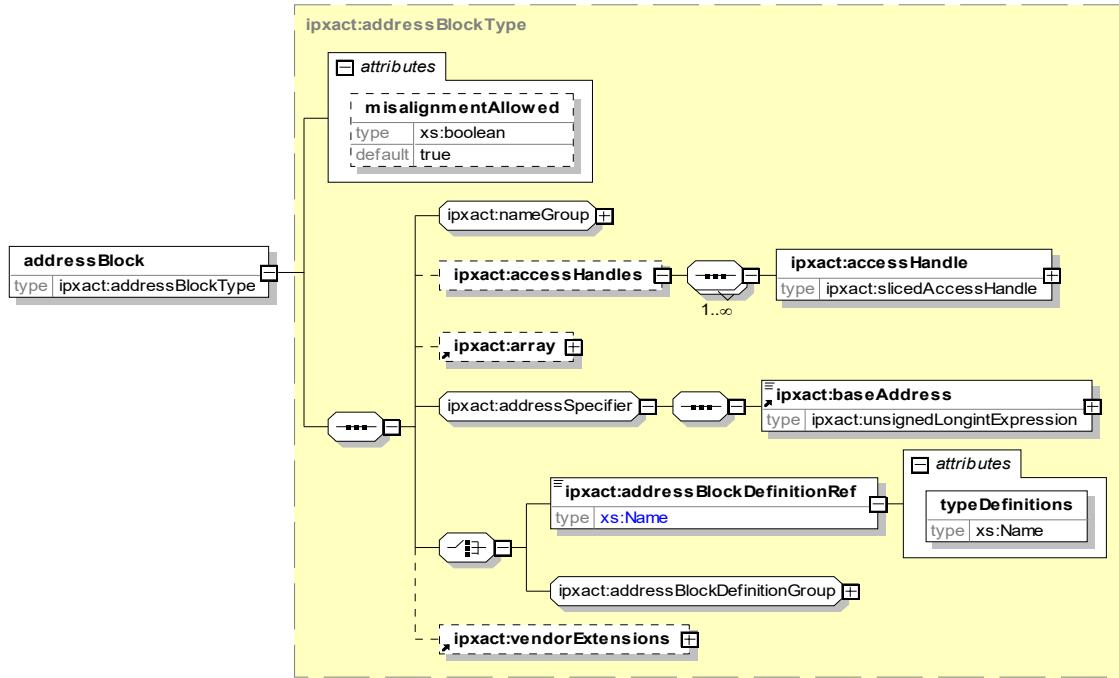
A memory map can be defined for each target interface of a component. The **memoryMaps** element contains an unbounded list of **memoryMap** elements. The **memoryMap** elements are referenced by the component's target interface. **memoryMap** contains the following mandatory and optional elements:

- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **memoryMap** element.
- b) A **memoryMap** element contains either a **memoryMapDefinitionRef** element or **memoryMapGroup** elements.
 - 1) **memoryMapDefinitionRef** (mandatory) is a reference to a **memoryMapDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element.
 - 2) **memoryMapGroup** (optional) is any number of the following:
 - i) **addressBlock** describes a single block. See [6.12.2](#).
 - ii) **bank** represents a collection of address blocks, banks, or subspace maps. See [6.12.5](#).
 - iii) **subspaceMap** maps the address subspaces of initiator interfaces into the target's memory map. See [6.12.9](#).
- c) **memoryRemap** (optional) element describes additional address blocks, banks, and subspace maps of a target bus interface in a specific mode.
- d) **addressUnitBits** element (optional; type: **unsignedPositiveLongintExpression**, see [C.7.14](#)) defines the number of data bits in each address increment of the memory map. This is required to allow the elements in the memory map to define items such as register offsets. If this element is not present, it is presumed to be 8.
- e) The optional **shared** element (default: **undefined**) defines the sharing properties of the memory map. When the value is **yes**, the contents of the **memoryMap** are shared by all the references to this **memoryMap**; when the value is **no**, the contents of the **memoryMap** are not shared; and when the value is **undefined**, the sharing of the **memoryMap** is undefined.
- f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the memory map. See [C.27](#).

6.12.2 Address block

6.12.2.1 Schema

The following schema details the information contained in the **addressBlock** element, which may appear in a **memoryMap** element. It is of type *addressBlockType*.



6.12.2.2 Description

The **addressBlock** element describes a single, contiguous block of memory that is part of a memory map. **addressBlock** contains the following mandatory and optional elements:

- misalignmentAllowed** (optional; type; *boolean*; default **true**). Indicates if register **addressOffset** may be misaligned in the **addressBlock**. See [SCR 7.27](#).
- nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **memoryMap**, **memoryRemap**, **localMemoryMap**, or **bank** elements.
- accessHandles** (optional) specifies view-dependent naming for this address block within the corresponding RTL. See [C.1.2](#).
- array** element has sub-elements **dim** (mandatory) and **stride** (optional). See [C.4.3](#).
- addressSpecifier** group includes the **baseAddress** (mandatory; type *unsignedLongintExpression*, see [C.7.12](#)), which specifies the starting address of the block. The **baseAddress** is expressed in addressing units from the containing element's **addressUnitBits** element.
- An **addressBlock** element contains either an **addressBlockDefinitionRef** element or **addressBlockDefinitionGroup** elements.
 - addressBlockDefinitionRef** (mandatory) is a reference to an **addressBlockDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element. See [6.12.3](#).
 - addressBlockDefinitionGroup** group describes the definition information about address blocks. It contains mandatory and optional elements including the **range** element. See [6.12.3](#).

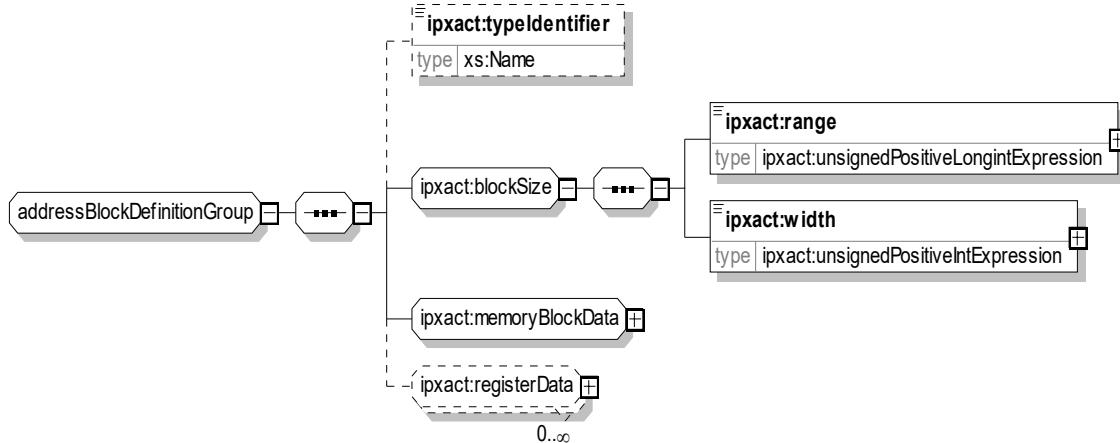
- g) **vendorExtensions** (optional) adds any extra vendor-specific data related to the address block. See [C.27](#).

See also [SCR 8.1](#), [SCR 8.4](#), and [SCR 7.27](#).

6.12.3 Address block definition group

6.12.3.1 Schema

The following schema details the information contained in the **addressBlockDefinitionGroup** group, which may appear in an **addressBlock** element.



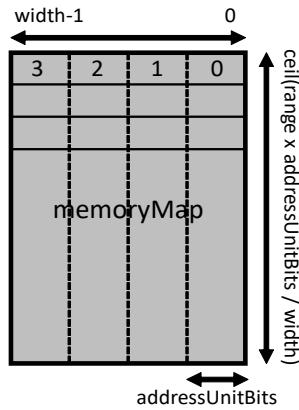
6.12.3.2 Description

The **addressBlockDefinitionGroup** group describes the definition information about address blocks. It contains the following mandatory and optional elements:

- typeIdentifier** (optional; type: *Name*) indicates multiple address block elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in the **addressBlockDefinitionGroup**.
- blockSize** group includes the following:
 - range** (mandatory; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) gives the address range of an address block. This is expressed as the number of addressable units. The size of an addressable unit is defined inside the containing element's **addressUnitBits** element.
 - width** (mandatory; type: *unsignedPositiveIntExpression*, see [C.7.11](#)) is the bit width of a row in the address block. A row in an address block sets the maximum single transfer size into the memory map allowed by the referencing bus interface and also defines the maximum size that a single register can be defined across an interconnection.
- memoryBlockData** group contains information about usage, access, volatility, and other parameters. See [6.12.4](#).
- registerData** group contains information about the grouping of bits into registers and fields. See [6.14.1](#).

The **range** and **width** elements are related by the following formulas:

$$\begin{aligned}
 \text{number_of_bits_in_block} &= \text{addressUnitBits} \times \text{range} \\
 \text{number_of_rows_in_block} &= \text{ceil}(\text{number_of_bits_in_block} / \text{width})
 \end{aligned}$$

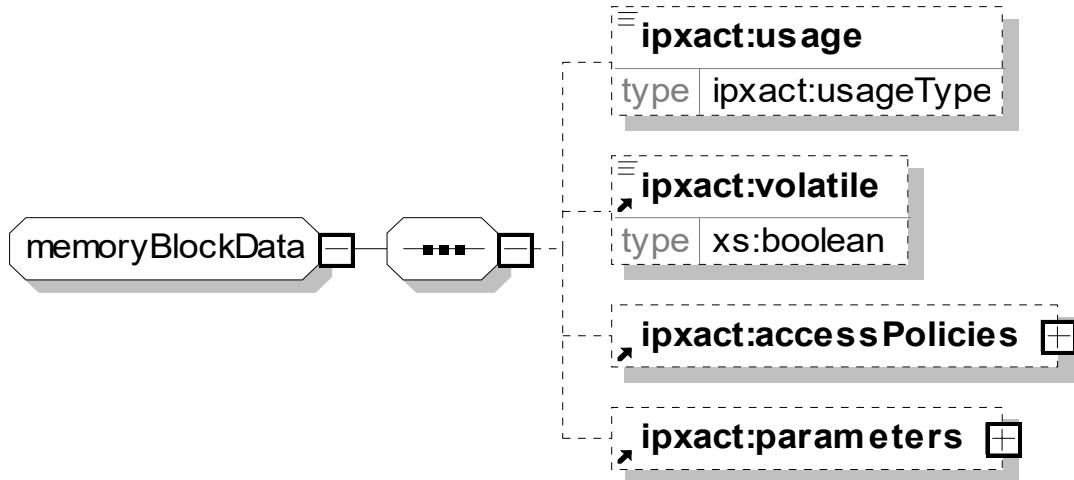


See also [SCR 8.1](#) and [SCR 7.15](#).

6.12.4 memoryBlockData group

6.12.4.1 Schema

The following schema details the information contained in the **memoryBlockData** group, an optional part of both **addressBlock** and **bank**.



6.12.4.2 Description

The **memoryBlockData** group is a collection of elements that contains further specification of **addressBlock** or **bank** elements. It contains the following elements:

- usage** (optional) specifies the type of usage for the address block or bank to which it belongs.
 - For an **addressBlock**:
 - memory** defines, when the **access** element inside an **accessPolicy** element is set to **read-only**, the entire range of the **addressBlock** as a read-only memory (ROM). If the **access** element is set to **read-write**, the entire range of the **addressBlock** is a random access

memory (RAM). If the **access** element is set to **write-only**, the entire range of the **addressBlock** is a write-only memory. This usage type can contain virtual registers.

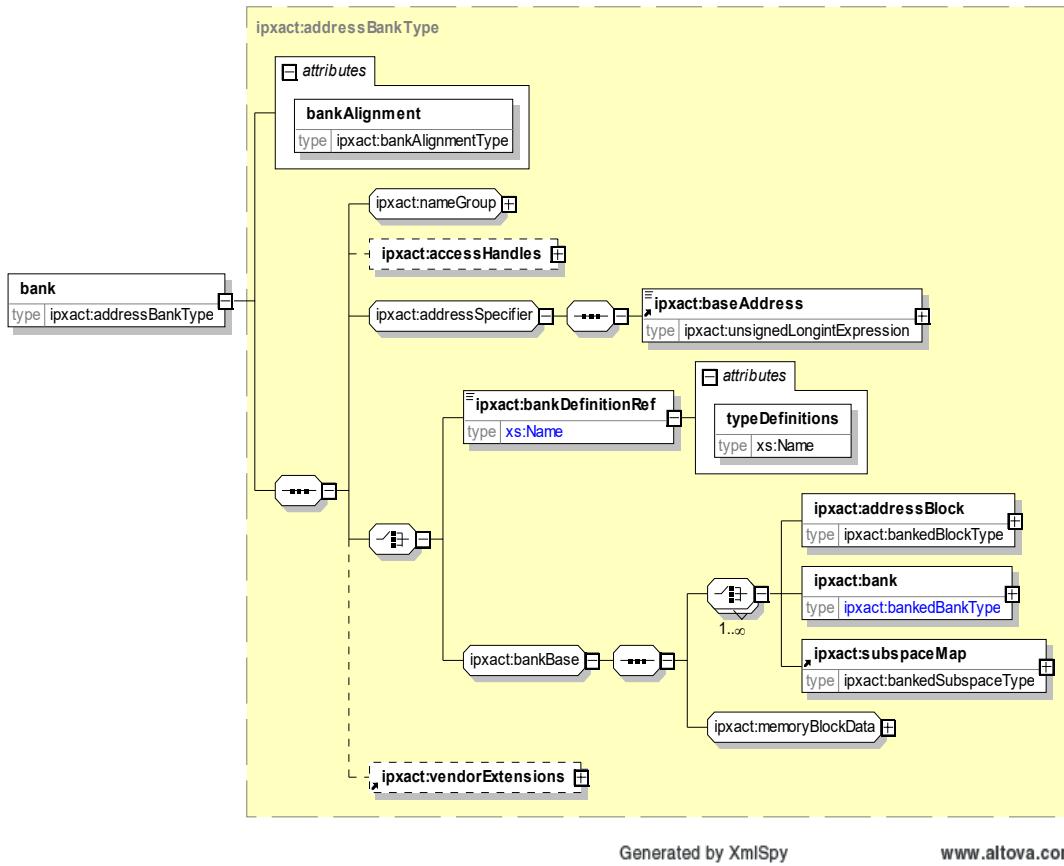
- ii) **register** defines the entire range of the **addressBlock** as possible locations for registers.
 - iii) **reserved** defines the entire range of the **addressBlock** as reserved or for unknown usage to IP-XACT. This type shall not contain registers.
 - iv) If unspecified, the presumed value for **usage** shall be **register** if the **addressBlock** contains **register** elements; otherwise it is **reserved**.
- 5) For a **bank**:
- i) **memory** defines all containing **addressBlock** elements are of this usage type.
 - ii) **register** defines all containing **addressBlock** elements are of this usage type.
 - iii) **reserved** defines all containing **addressBlock** elements are of this usage type.
 - iv) Unspecified usage means the bank may contain a mixture of **memory**, **register**, and **reserved addressBlock** elements.
- b) **volatile** (optional; type: *boolean*) when **true** indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. This element implies there is some additional mechanism by which these registers can acquire new values other than reads/writes/resets and other access methods known to IP-XACT. If this element is not present, it is presumed to be **false** for a **field** and unspecified for a **bank**, **addressBlock**, or **register**.
- c) **accessPolicies** (optional) specifies the accessibility of data in the address block or bank for different operating modes. See [C.2](#) and [C.3](#).
- d) **parameters** (optional) details any additional parameters that describe the address block or bank for generator usage. See [C.21](#).

See also [SCR 8.3](#) and [SCR 8.5](#).

6.12.5 Bank

6.12.5.1 Schema

The following schema details the information contained in the **bank** element, which can appear in a **memoryMap** element. It is of type **addressBankType**.



6.12.5.2 Description

The **bank** element allows multiple **addressBlocks**, **banks**, or **subspaceMaps** to be concatenated together horizontally or vertically as a single entity. It contains the following attributes and elements:

- bankAlignment** (mandatory) attribute organizes the bank:
 - parallel** specifies each item is located at the same base address with different bit offsets. The bit offset of the first item in the bank always starts at 0, the offset of the next items in the bank is equal to the widths of all the previous items.
 - serial** specifies the first item is located at the bank's base address. Each subsequent item is located at the previous item's address, plus the range of that item (adjusted for LAU and bus width considerations, rounded up to the next whole multiple). This allows the user to specify only a single base address for the bank and have each item assigned an address in sequence.
- nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **memoryMap**, **memoryRemap**, and **localMemoryMap** elements.
- accessHandles** (optional) specifies view-dependent naming for this bank within the corresponding view. See [C.1.1](#).

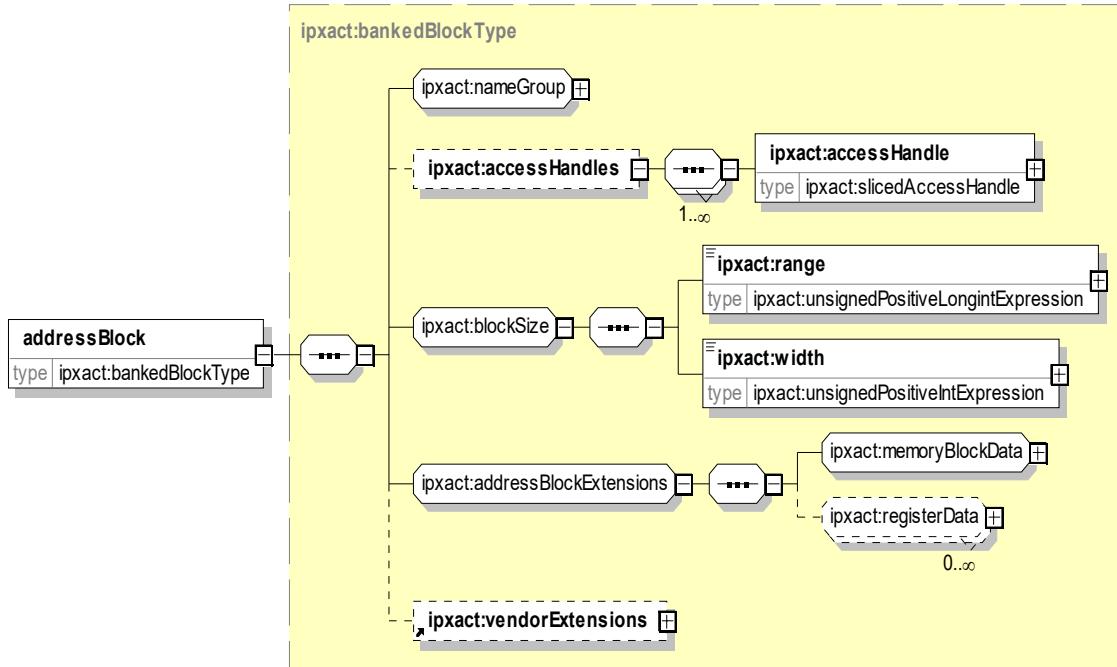
- d) **addressSpecifier** group includes the following:
 - baseAddress** (mandatory; type: *unsignedLongintExpression*, see [C.7.12](#)) specifies the starting address of the block. The **baseAddress** is expressed in addressing units from the containing element's **addressUnitBits** element.
- e) A **bank** element contains either a **bankDefinitionRef** element or **bankBase** group elements.
 - 1) **bankDefinitionRef** (mandatory) is a reference to a **bankDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element.
 - 2) **bankBase** is later used inside the **bankedBaseType** type to create recursion. A **bankBase** group also contains at least one of the following:
 - i) **addressBlock** is an address block that makes up part of the bank. See [6.12.6](#).
 - ii) **bank** is a bank within the bank. This allows for complex configurations with nested banks. See [6.12.7](#).
 - iii) **subspaceMap** is a reference to the initiator's address map for inclusion in the bank. See [6.12.9](#).
 - iv) **memoryBlockData** (mandatory) is a collection of elements that contains further specification of addressBlock or bank elements See [6.12.4](#).
- f) **vendorExtensions** adds any extra vendor-specific data related to this bank. See [C.27](#).

See also [SCR 8.2](#).

6.12.6 Banked address block

6.12.6.1 Schema

The following schema details the information contained in the **addressBlock** element, which can appear in a **bank** element. It is of type **bankedBlockType**.



6.12.6.2 Description

The **addressBlock** element inside a bank element describes a single, contiguous block of memory that is part of a bank. **addressBlock** contains the following mandatory and optional elements:

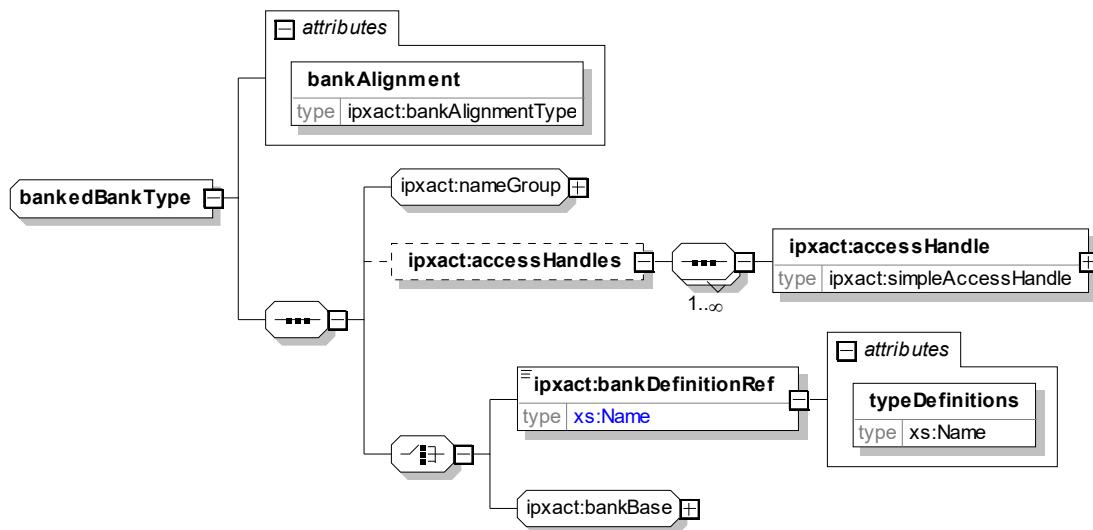
- a) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **bank** element.
- b) **accessHandles** (optional) specifies view-dependent naming for this address block within the corresponding RTL. See [C.1.2](#).
- c) **blockSize** group includes the following:
 - 1) **range** (mandatory; type: *unsignedPositiveLongintExpression*) gives the address range of an address space. See [C.7.14](#).
 - 2) **width** (mandatory) is the bit width of a row in the address space. The type of this element is set to **nonNegativeInteger**. The **width** element is of type *unsignedIntExpression*. See [C.7.11](#).
- d) **addressBlockExtensions** (mandatory) contains **memoryBlockData** (see [6.12.4](#)) and may contain **registerData**. See [6.14.1](#).
 - 1) **memoryBlockData** group contains information about usage, access, volatility, and other parameters. See [6.12.4](#).
 - 2) **registerData** group contains information about the grouping of bits into registers and fields. See [6.14.1](#).
- e) **vendorExtensions** (optional) adds any extra vendor-specific data related to the address block. See [C.27](#).

See also [SCR 7.5](#).

6.12.7 Banked bank

6.12.7.1 Schema

The following schema details the information contained in the nested **bank** element, which can appear in another **bank** element. It is of type **bankedBankType**.



6.12.7.2 Description

The **bank** element within another bank element (banked bank) allows multiple address **blocks**, **banks**, or **subspaceMaps** to be concatenated together horizontally or vertically as a single entity. It contains the following attributes and elements:

- a) **bankAlignment** (mandatory) attribute organizes the bank:
 - 1) **parallel** specifies each item is located at the same base address with different bit offsets. The bit offset of the first item in the bank always starts at 0, the offset of the next items in the bank is equal to the widths of all the previous items.
 - 2) **serial** specifies the first item is located at the bank's base address. Each subsequent item is located at the previous item's address, plus the range of that item (adjusted for LAU and bus width considerations, rounded up to the next whole multiple). This allows the user to specify only a single base address for the bank and have each item assigned an address in sequence.
- b) **nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **bank** element.
- c) **accessHandles** (optional) specifies view-dependent naming for this bank within the corresponding RTL. See [C.1.1](#).
- d) A banked **bank** element contains either a **bankDefinitionRef** element or **bankBase** group elements.
 - 1) **bankDefinitionRef** (mandatory) is a reference to a **bankDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element.
 - 2) The **bank** element of type **bankedBankType** contains the **bankBase** group. This group is defined inside the **bank** element of type **addressBankType**. See [6.12.5](#). The effect of its inclusion here creates recursion, whereby banks may be included inside banks included inside banks.

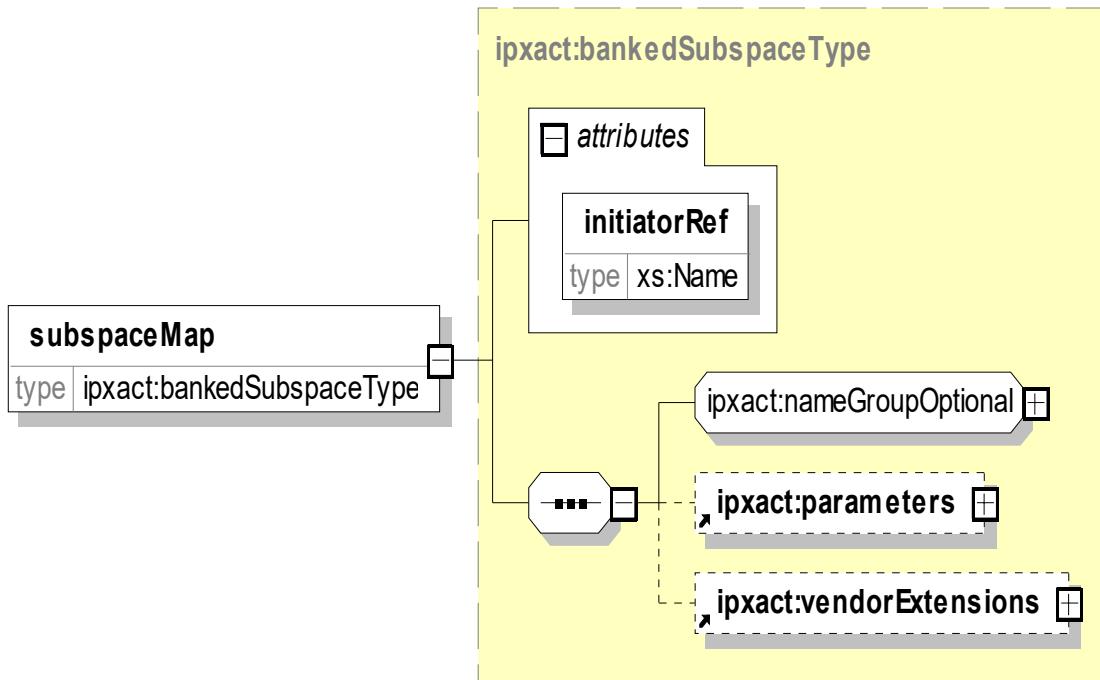
NOTE—A banked bank is similar to a bank in a memory map (see [6.12.5](#)); the only difference is there is no **baseAddress** element in a **bank** of type **bankedBankType**.

See also [SCR 8.2](#).

6.12.8 Banked subspace

6.12.8.1 Schema

The following schema details the information contained in the **subspaceMap** element, which can appear in a **bank** element. It is of type **bankSubspaceType**.



6.12.8.2 Description

The **subspaceMap** element, within an address bank, allows a bank to map the address space of an initiator interface into the bank. It contains the following elements:

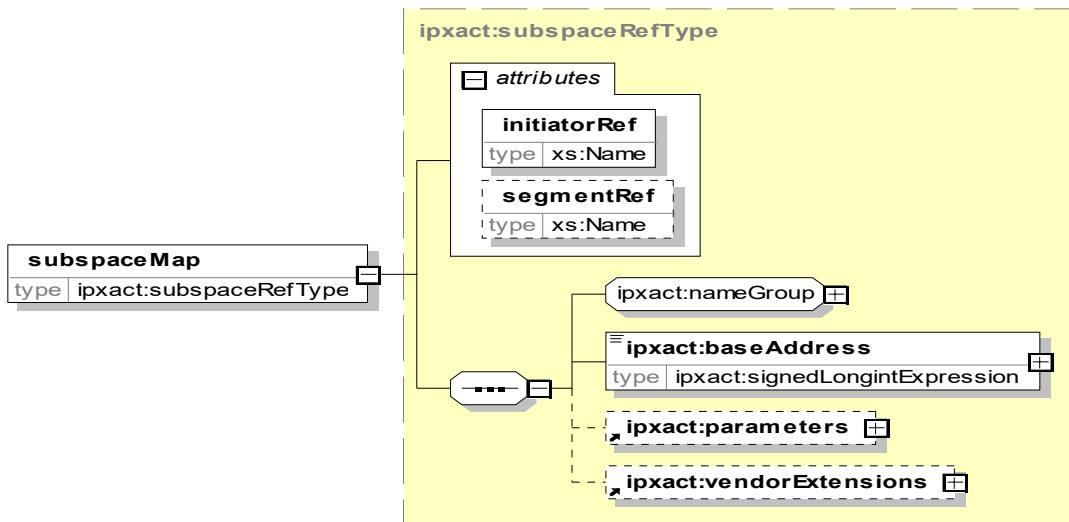
- initiatorRef** attribute (mandatory; type: *Name*) contains the name of the initiator interface whose address space needs to be mapped. This shall reference a bus interface name with an interface mode of initiator (see [6.7.5](#)).
- nameGroupOptional** group is defined in [C.19.2](#). The **name** of the **subspaceMap** shall be unique within the containing **bank** element.
- parameters** details any additional parameters that apply to the **subspaceMap**. See [C.21](#).
- vendorExtensions** adds any extra vendor-specific data related to the **subspaceMap**. See [C.27](#).

See also [SCR 8.2](#).

6.12.9 Subspace map

6.12.9.1 Schema

The following schema details the information contained in the **subspaceMap** element, which can appear in a **memoryMap** element. It is of type *subspaceRefType*.



6.12.9.2 Description

The **subspaceMap** element maps the address space of an initiator interface from an opaque bus bridge into the memory map. It contains the following elements and attributes:

- initiatorRef** (mandatory; type: *Name*) attribute contains the name of the initiator interface whose address space needs to be mapped. This shall reference a bus interface name with an interface mode of initiator (see [6.7.5](#)).
- segmentRef** (optional; type: *Name*) attribute references a **segment** in the **addressSpace** referred by the **initiatorRef** attribute. If the **segmentRef** attribute is not present, the entire **addressSpace** is presumed to be referenced.
- nameGroup** group is defined in [C.19](#). The **name** of the **subspaceMap** shall be unique within the containing **memoryMap** or **memoryRemap** element.
- baseAddress** (mandatory; type: *signedLongintExpression*, see [C.7.12](#)) specifies the starting address of the block. The **baseAddress** is expressed in addressing units from the containing element's **addressUnitBits** element.
- parameters** (optional) details any additional parameters that apply to the **subspaceMap**. See [C.21](#).
- vendorExtensions** (optional) adds any extra vendor-specific data related to the **subspaceMap**. See [C.27](#).

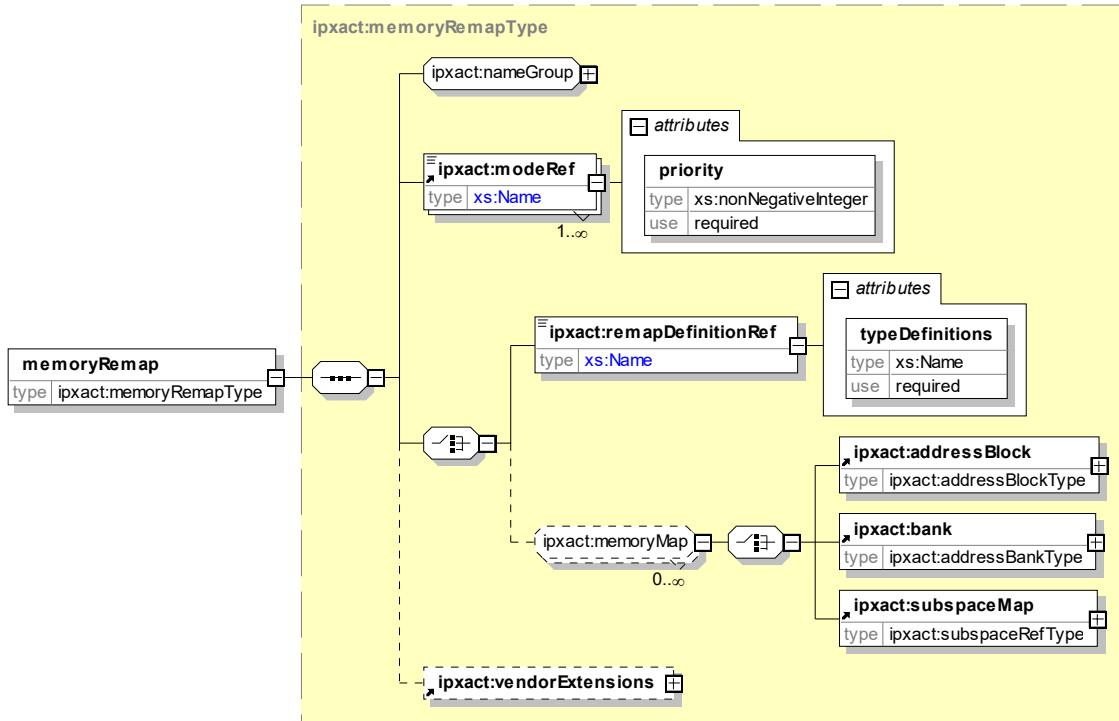
See also [SCR 3.17](#), [SCR 9.7](#), and [SCR 15.1](#).

6.13 Remapping

6.13.1 Memory remap

6.13.1.1 Schema

The following schema details the information contained in the **memoryRemap** element, which can appear in a **memoryMap** element. It is of type **memoryRemapType**.



6.13.1.2 Description

The **memoryRemap** element describes additional **addressBlocks**, **banks**, and **subspaceMaps** that are mapped on the referencing target bus interface in specific modes. If multiple **memoryRemap** elements are active, then the ones with highest priority are added to the **memoryMap**. This element contains the following elements, attributes, and groups:

- nameGroup** group is defined in [C.19](#). The **name** of the **memoryRemap** shall be unique within the containing **memoryMap** element.
- modeRef** (mandatory) identifies the operating **mode** for which the **memoryRemap** element is active by referencing a mode name in the containing description and providing a priority. A **memoryRemap** is active if it references an active mode and that has highest priority among the active modes referenced in all **memoryRemap** elements of the **memoryMap**. The **modeRef** element value shall be unique within the containing **memoryMap** element. The **modeRef** element has an attribute **priority** (mandatory; type **nonNegativeInteger**) which indicates the priority of the referenced mode for this **memoryRemap**. The lower the value, the higher the priority. The priority value of a **modeRef** element is not necessarily unique in the scope of the referenced modes inside the **memoryMap** element. Hence, multiple **memoryRemap** elements can be active.
- A **memoryRemap** element contains either a **remapDefinitionRef** element or **memoryMap** group elements.

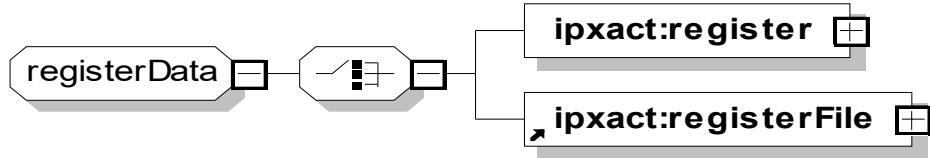
- 1) **remapDefinitionRef** (mandatory) is a reference to a **memoryRemapDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element.
- 2) **memoryMap** group (optional) is any number of the following:
 - i) **addressBlock** describes a single block. See [6.12.2](#).
 - ii) **bank** represents a collection of address blocks, banks, or subspace maps. See [6.12.5](#).
 - iii) **subspaceMap** maps the address subspaces of initiator interfaces into the target's memory map. See [6.12.9](#).

6.14 Registers

6.14.1 Register data

6.14.1.1 Schema

The following schema details the information contained in the **registerData** group that may appear as an element inside the **addressBlock** element.



6.14.1.2 Description

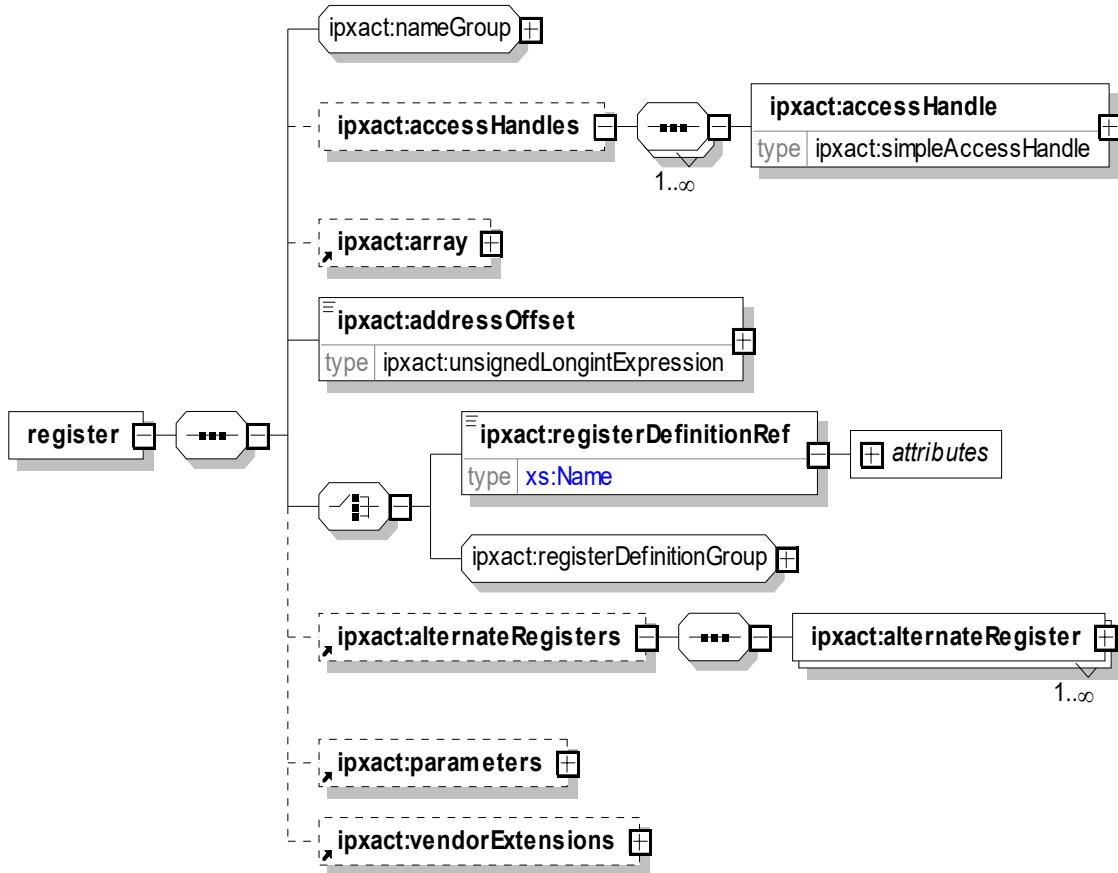
The **registerData** group describes registers and register files. The containing **register/name** elements, the **register/alternateRegister/name** elements, and the **registerFile/name** elements shall be unique within the containing **addressBlock** element. The **registerData** group contains one of the following elements:

- a) **register** defines a list of registers contained in this **addressBlock**. See [6.14.2](#).
- b) **registerFile** defines a list of register files contained in this **addressBlock**. See [6.14.6](#).

6.14.2 Register

6.14.2.1 Schema

The following schema details the information contained in the **register** element, which is contained in the **registerData** group that may appear as an element inside the **addressBlock** element which is part of the higher-level **memoryMap**, **memoryRemap**, **localMemoryMap**, and **bank** elements. This element describes a **register**.



6.14.2.2 Description

A **register** element describes a register in an address block or register file. The bits in the register are numbered from `size-1` down to 0, with bit zero (0) being the least significant bit. **register** contains the following elements:

- nameGroup** group is defined in [C.19](#). The register **name** shall be unique in the containing **addressBlock** or **registerFile** element.
- accessHandles** (optional) specifies view-dependent naming for this register within the corresponding RTL. See [C.1.1](#).
- array** element has sub-elements **dim** (mandatory) and **stride** (optional) as defined in [C.4.3](#).
- addressOffset** (mandatory; type: `unsignedLongintExpression`, see [C.7.12](#)) describes the offset from the start of the containing **addressBlock** or **registerFile** element. The **addressOffset** is expressed in addressing units from the containing element's **addressUnitBits** element.
- A **register** element contains either a **registerDefinitionRef** element or **registerDefinitionGroup** elements.
 - registerDefinitionRef** (mandatory) is a reference to a **registerDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element.
 - registerDefinitionGroup** group describes additional elements for a register. See [6.14.3](#).
- alternateRegisters** (optional) describes alternate descriptions for the containing register. See [6.14.4](#).

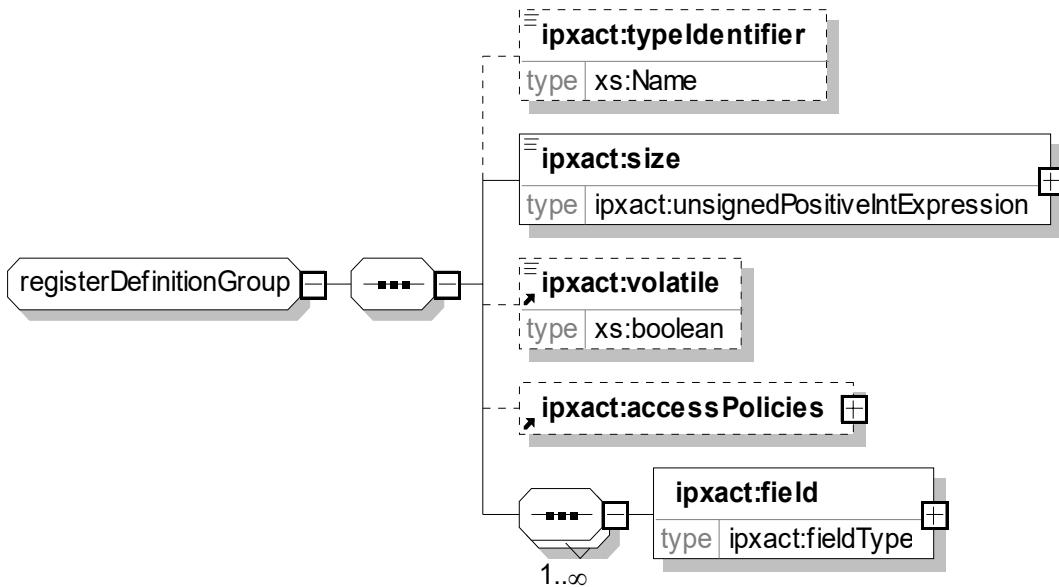
- g) **parameters** (optional) describes any parameter names and types when the register width can be parameterized. See [C.21](#).
- h) **vendorExtensions** (optional) adds any extra vendor-specific data related to this register. See [C.27](#).

See also [SCR 7.1](#), [SCR 7.2](#), [SCR 7.3](#), [SCR 7.4](#), [SCR 7.5](#), [SCR 7.7](#), [SCR 7.8](#), [SCR 7.9](#), [SCR 7.13](#), and [SCR 8.3](#).

6.14.3 Register definition group

6.14.3.1 Schema

The following schema details the information contained in the *registerDefinitionGroup* group, which is contained in the **register** element. This group describes register definition information.



6.14.3.2 Description

A *registerDefinitionGroup* group contains the following elements:

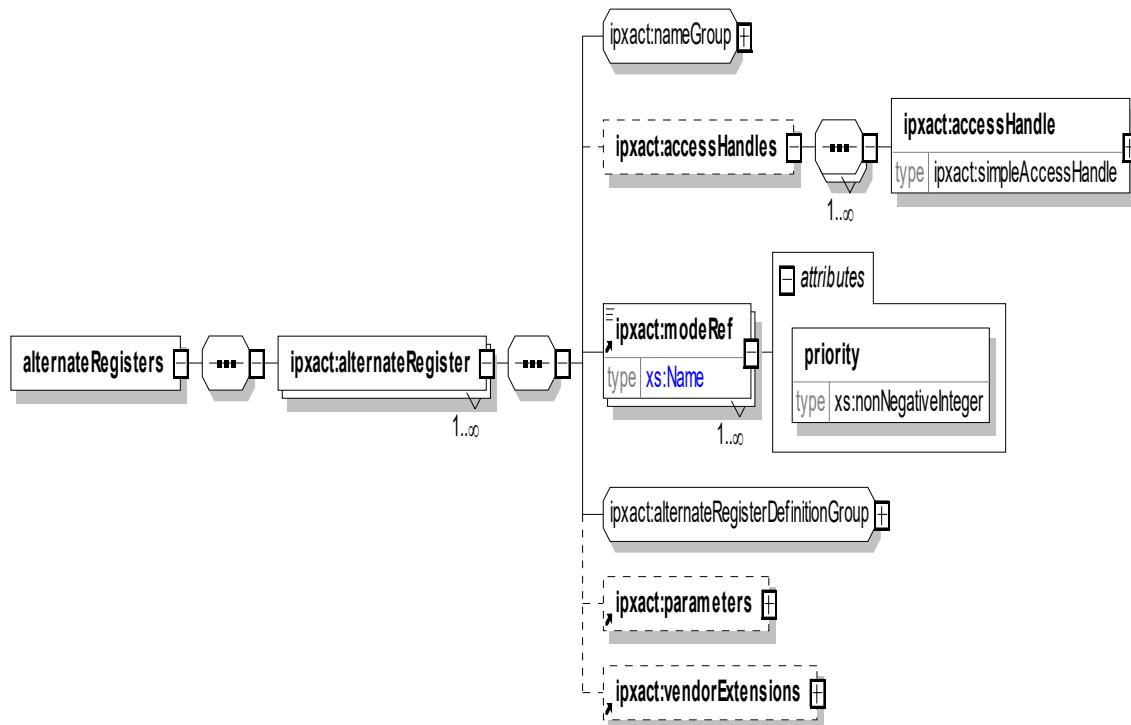
- a) **typeIdentifier** (optional) indicates multiple register elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in the **registerDefinitionGroup**.
- b) **size** (mandatory; type: *unsignedPositiveIntExpression*, see [C.7.14](#)) is the width of the register, counting in bits.
- c) **volatile** (optional; type: *boolean*) when **true** indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this register can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. If this element is not present, no presumptions can be made about its value.
- d) **accessPolicies** (optional) describes the register access for different operating modes. See [C.2](#) and [C.3](#).
- e) **field** (mandatory) describes any bit fields in a register. See [6.14.8](#).

See also [SCR 7.1](#), [SCR 7.2](#), [SCR 7.3](#), [SCR 7.4](#), [SCR 7.5](#), [SCR 7.7](#), [SCR 7.8](#), [SCR 7.9](#), [SCR 7.13](#), and [SCR 8.3](#).

6.14.4 Alternate registers

6.14.4.1 Schema

The following schema details the information contained in the **alternateRegister** element. Each **alternateRegister** element contained within the same **alternateRegisters** element provides an alternate description for the containing register element.



6.14.4.2 Description

An **alternateRegister** element contains an alternate definition for the containing register. **alternateRegister** contains the following elements:

- nameGroup** group is defined in [C.19](#). The **alternateRegister** **name** shall be unique in the containing **register** element.
- accessHandles** (optional) specifies view-dependent naming for this register within the corresponding RTL. See [C.1.1](#).
- modeRef** (mandatory) identifies the operating **mode** for which the **alternateRegister** element is active by referencing a mode name in the containing description and providing a priority. An **alternateRegister** is active if it references an active mode and that has highest priority among the active modes referenced in all **alternateRegister** elements of the **register**. The **modeRef** element value shall be unique within the containing **register** element. The **modeRef** element has an attribute **priority** (mandatory; type *nonNegativeInteger*) which indicates the priority of the referenced mode for this **alternateRegister**. The lower the value, the higher the priority. The priority value of a **modeRef** element shall be unique in the scope of the referenced modes inside the **register** element. **alternateRegisterDefinitionGroup** group describes additional elements for an alternate register. See [6.14.5](#).

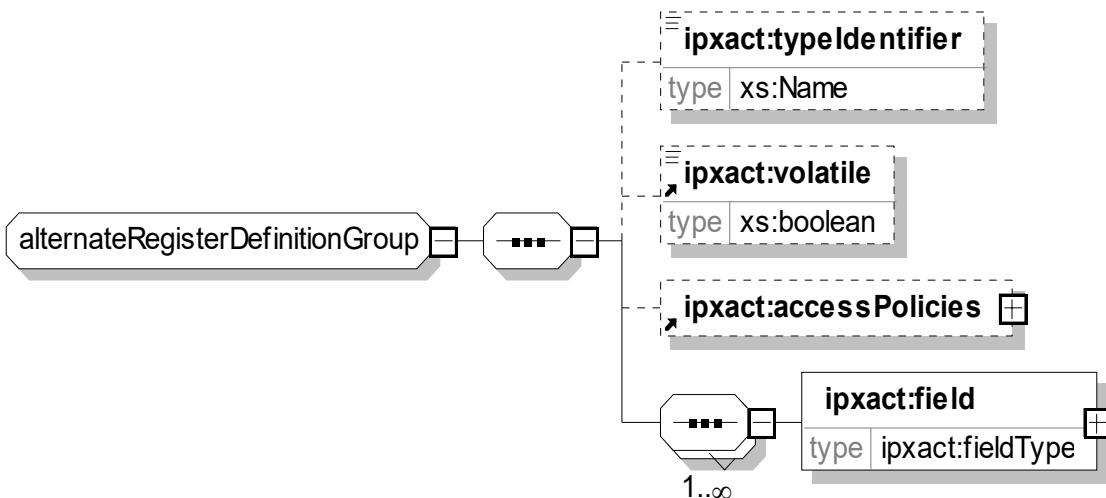
- d) **parameters** (optional) describes any parameter names and types when the register width can be parameterized. See [C.21](#).
- e) **vendorExtensions** (optional) adds any extra vendor-specific data related to this register. See [C.27](#).

See also [SCR 7.1](#), [SCR 7.2](#), [SCR 7.3](#), [SCR 7.4](#), [SCR 7.7](#), [SCR 7.8](#), [SCR 7.9](#), [SCR 7.13](#), [SCR 8.3](#), and [SCR 8.6](#).

6.14.5 Alternate register definition group

6.14.5.1 Schema

The following schema details the information contained in the **alternateRegisterDefinitionGroup** group, which is contained in the **alternateRegister** element. This group describes alternate register definition information.



6.14.5.2 Description

An **alternateRegisterDefinitionGroup** group contains the following elements:

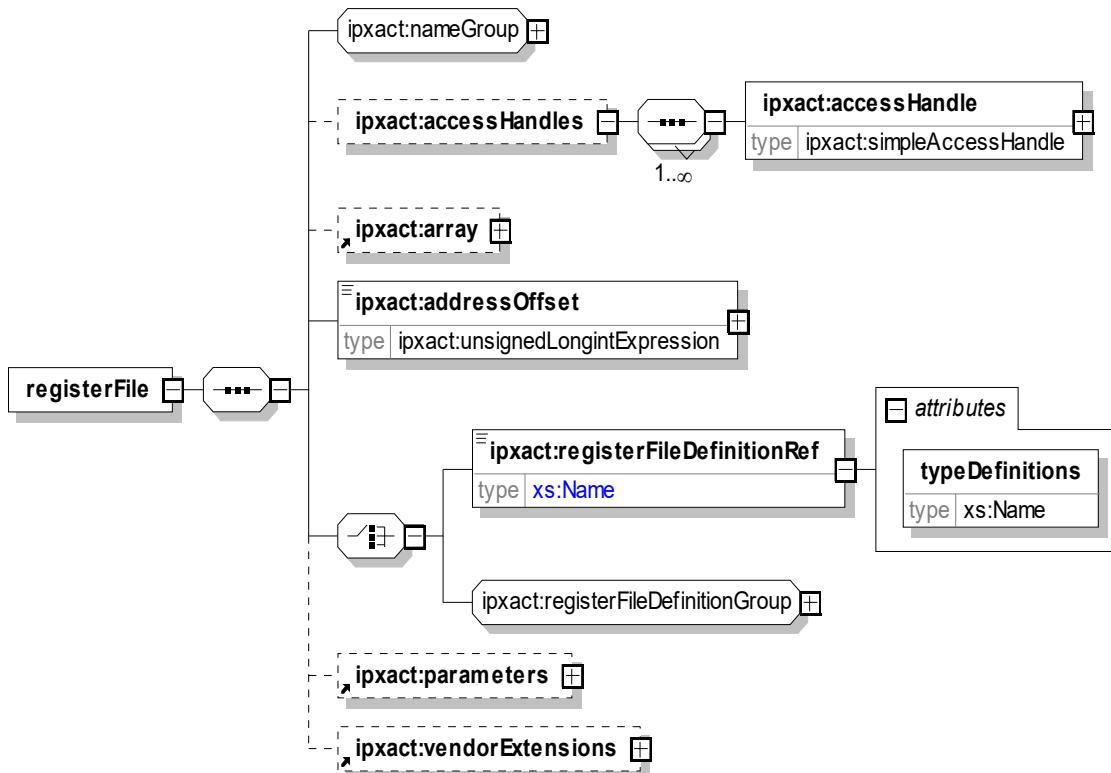
- a) **typeIdentifier** (optional; type: *Name*) indicates multiple register elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in the **alternateRegisterDefinitionGroup**.
- b) **volatile** (optional; type: *boolean*) when **true** indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this register can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. If this element is not present, no presumptions can be made about its value.
- c) **accessPolicies** (optional) describes the alternate register access for different operating modes (see [C.2](#) and [C.3](#)).
- d) **field** (required) describes any bit fields in a register. See [6.14.8](#).

See also [SCR 7.1](#), [SCR 7.2](#), [SCR 7.4](#), [SCR 7.5](#), [SCR 7.8](#), [SCR 7.9](#), [SCR 7.13](#), and [SCR 8.6](#).

6.14.6 Register file

6.14.6.1 Schema

The following schema details the information contained in the **registerFile** element, which is contained in the **registerData** group that may appear as an element inside the **addressBlock** element. This element describes a register file.



6.14.6.2 Description

A **registerFile** element describes a grouping of registers in an address block or register file. **registerFile** contains the following elements:

- nameGroup** group is defined in [C.19](#). The **registerFile** name shall be unique in the containing **addressBlock** or **registerFile** element.
- accessHandles** (optional) specifies view-dependent naming for this register files within the corresponding RTL. See [C.1.1](#).
- array** element has sub-elements **dim** (mandatory) and **stride** (optional) as defined in [C.29](#).
- addressOffset** (mandatory; type: *unsignedLongintExpression*, see [C.7.12](#)) describes the offset from the start of the containing **addressBlock** or **registerFile** element. The **addressOffset** is expressed in addressing units from the containing element's **addressUnitBits** element.
- A **registerFile** element contains either a **registerFileDefinitionRef** element or **registerFileDefinitionGroup** elements.
 - registerFileDefinitionRef** (mandatory) is a reference to a **registerFileDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element.
 - registerFileDefinitionGroup** group describes additional elements for a register file. See [6.14.7](#).

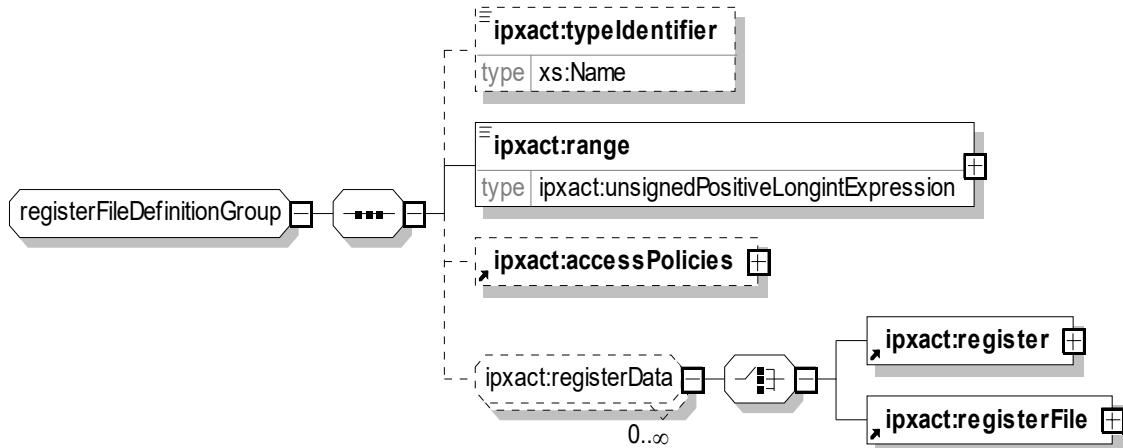
- f) **parameters** (optional) describes any parameter names and types when the register width can be parameterized. See [C.21](#).
- g) **vendorExtensions** (optional) adds any extra vendor-specific data related to this register. See [C.27](#).

See also [SCR 7.6](#), [SCR 7.7](#), and [SCR 7.14](#).

6.14.7 Register file definition group

6.14.7.1 Schema

The following schema details the information contained in the **registerFileDefinitionGroup** group, which may appear as an element inside the **registerFile** element.



6.14.7.2 Description

The **registerFileDefinitionGroup** group describes additional information for a register file.

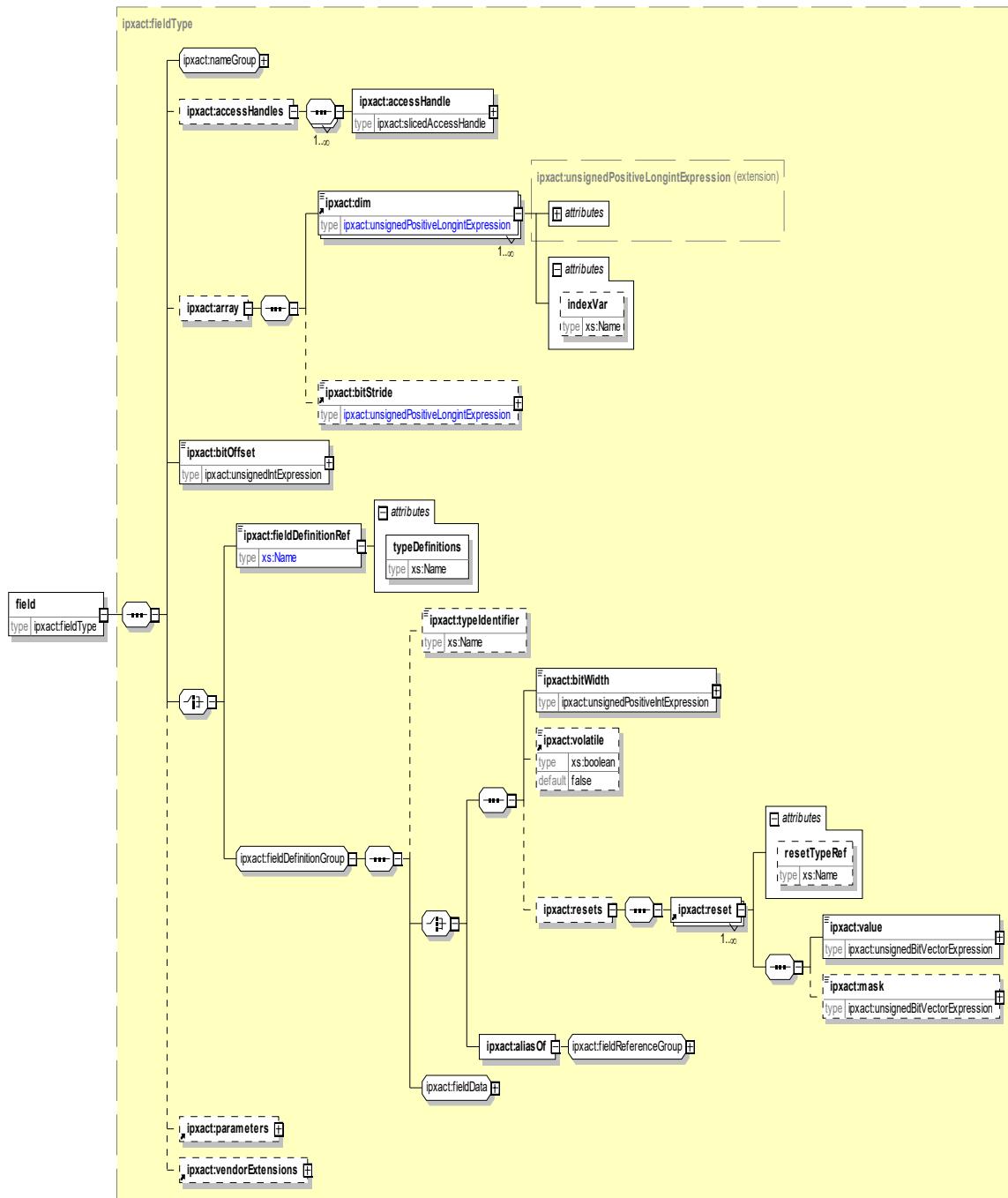
- a) **typeIdentifier** (optional; type: *Name*) indicates multiple register elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in the **registerDefinitionGroup**.
- b) **range** (mandatory; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) gives the range of a register file. This is expressed as the number of addressable units of the register file. The size of an addressable unit is defined inside the containing element's **addressUnitBits** element.
- c) **accessPolicies** (optional) describes the access policies of the register file. See [C.2](#) and [C.3](#).
- d) **registerData** group contains information about the grouping of bits into registers and fields. See [6.14.1](#).

See also [SCR 7.14](#).

6.14.8 Register bit fields

6.14.8.1 Schema

The following schema details the information contained in the field element, which may appear as an element inside the register element. The register element is contained in the registerData group that may appear as an element inside the addressBlock element which is part of the higher-level **memoryMap**, **memoryRemap**, **localMemoryMap**, and **bank** elements. The field element describes a bit field of a register.



6.14.8.2 Description

A **field** element of a **register** describes a smaller bit field of a register. **field** contains the following elements:

- a) *nameGroup* group defines any descriptive text for the containing element. See [C.11](#).
 - b) **accessHandles** (optional) specifies view-dependent naming for this field within the corresponding RTL. See [C.1.2](#).

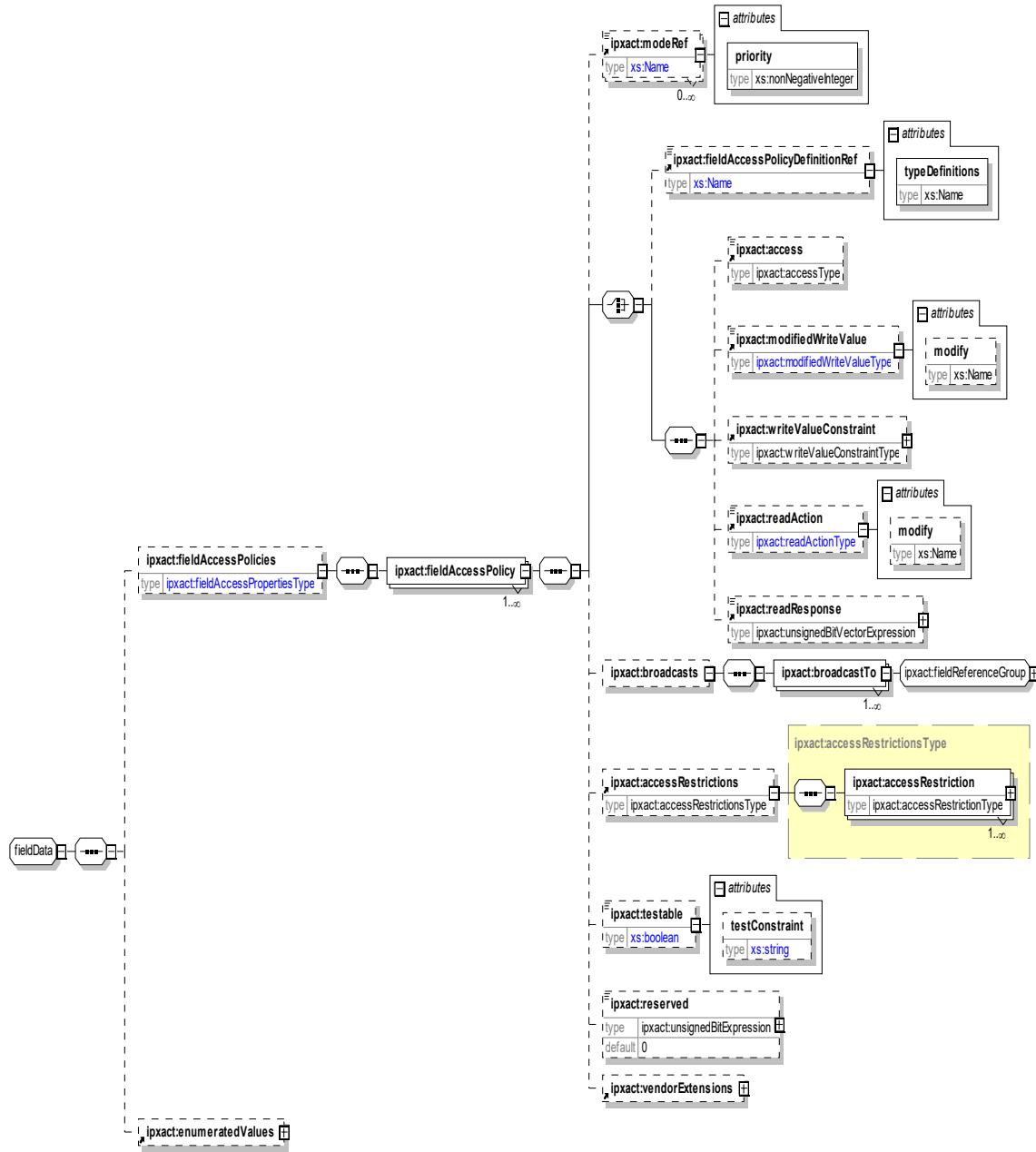
- c) **array** element has sub-elements **dim** (mandatory) and **bitStride** (optional). The differences with address block, register file, and register array are that stride is replaced by bitStride, which is expressed in bits. See [C.4.4](#).
- d) **bitOffset** (mandatory; type: *unsignedIntExpression*, see [C.7.11](#)) describes the offset (from bit 0 of the register) where this bit field starts.
- e) A **fieldElement** element contains either a **fieldDefinitionRef** element or **fieldDefinitionGroup** elements.
 - 1) **fieldDefinitionRef** (mandatory) is a reference to a **fieldDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element.
 - 2) **fieldDefinitionGroup** group describes additional elements for a field.
 - i) **typeIdentifier** (optional; type: *Name*) indicates multiple fields elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in **fieldDefinitionGroup**.
 - ii) **bitWidth** (mandatory; type: *unsignedPositiveIntExpression*, see [C.7.13](#)) is the width of the field, counting in bits.
 - iii) **volatile** (optional; type: *boolean*; default: **false**) when **true** indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this field can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. If this element is not present, it is presumed to be **false**.
 - iv) **resets** (optional) describes the reset values of the field. Each reset element defines the reset value for a given reset type. reset has the following subelements:
 - The **resetTypeRef** attribute (optional; type: *Name*) identifies the type of reset being defined. If specified, this should correspond to a user-defined reset in the **resetTypes** element (see [6.21](#)). If not specified, the default reset type is HARD. The **resetTypeRef** attribute value shall be unique within the containing field element.
 - **value** (mandatory; type: *unsignedBitVectorExpression*, see [C.7.10](#)) contains the actual reset value.
 - **mask** (optional; type: *unsignedBitVectorExpression*, see [C.7.10](#)) defines which bits of the register field have a known reset value. A 1 bit in the mask means the corresponding bit of the register field has a known reset value; a 0 bit means it does not. All bits of the value that correspond to 0 bits of the mask are ignored. The absence of a mask element is equivalent to a mask of the same size as the register field consisting of all 1 bits.
 - v) **aliasOf** (optional) is a reference to another field in this component using a **fieldReferenceGroup**. This field is an alias of the referenced field meaning that the value of this field and the value of the referenced field are always identical. Field aliasing does not contribute to volatility of fields because the aliasing is explicitly described in IP-XACT. See [C.13](#).
 - vi) **fieldData** group describes additional elements for a field. See [6.14.9](#).
 - f) **parameters** (optional) details any additional parameters that describe the **field** for generator usage. See [C.21](#).
 - g) **vendorExtensions** (optional) adds any extra vendor-specific data related to this field. See [C.27](#).

See also [SCR 7.2](#), [SCR 7.4](#), [SCR 7.9](#), [SCR 7.10](#), [SCR 7.11](#), [SCR 7.12](#), [SCR 7.20](#), and [SCR 8.7](#).

6.14.9 Field data group

6.14.9.1 Schema

The following schema details the information contained in the **fieldData** group, which is contained inside the **field** element. This group describes the optional properties of the **fieldData** group definition.



6.14.9.2 Description

The **fieldData** group contains the following elements:

- a) **fieldAccessPolicy** describes field access properties for different operating modes:
 - 1) **modeRef** (optional) identifies the operating **mode** for which the **fieldAccessPolicy** element is active by referencing a mode name in the containing description and providing a priority. An **fieldAccessPolicy** is active if it references an active mode and that has highest priority among the active modes referenced in all **fieldAccessPolicy** elements of the **fieldAccessPolicies** element. The **modeRef** element value shall be unique within the containing **fieldAccessPolicies** element. The **modeRef** element has an attribute **priority** (mandatory; type *nonNegativeInteger*) which indicates the priority of the referenced mode for this **fieldAccessPolicy**. The lower the value, the higher the priority. The priority value of a modeRef element shall be unique in the scope of the referenced modes inside the **fieldAccessPolicies** element. If no **modeRef** element is present, then the **fieldAccessPolicy** element applies to all modes.
 - 2) A **fieldDataGroup** contains either a **fieldAccessPolicyDefinitionRef** element or **access**, **modifiedWriteValue**, **writeValueConstraint**, **readAction**, and **response** elements.
 - i) **fieldAccessPolicyDefinitionRef** (optional) is a reference to a **fieldAccessPolicyDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element.
 - ii) **access** (optional) indicates the accessibility of the field. If this is not present, the access defaults to read-write. This element can take one of several values. See [C.2](#).
 - iii) **modifiedWriteValue** (optional) element to describe the manipulation of data written to a field. The value shall be one of **oneToClear**, **oneToSet**, **oneToToggle**, **zeroToClear**, **zeroToSet**, **zeroToToggle**, **clear**, **set**, or **modify**. If the modifiedWriteValue element is not specified, the value written to the field is the value stored in the field.
 - oneToClear** means in a bitwise fashion each write data bit of a one shall clear (set to zero) the corresponding bit in the field, and each write data bit of a zero shall not effect that bit.
 - oneToClear** means in a bitwise fashion each write data bit of a one shall clear (set to zero) the corresponding bit in the field, and each write data bit of a zero shall not effect that bit.
 - oneToSet** means in a bitwise fashion each write data bit of a one shall set (set to one) the corresponding bit in the field, and each write data bit of a zero shall not effect that bit.
 - oneToToggle** means in a bitwise fashion each write data bit of a one shall toggle the corresponding bit in the field, and each write data bit of a zero shall not effect that bit.
 - zeroToClear** means in a bitwise fashion each write data bit of a zero shall clear (set to zero) the corresponding bit in the field, and each write data bit of a one shall not effect that bit.
 - zeroToSet** means in a bitwise fashion each write data bit of a zero shall set (set to one) the corresponding bit in the field, and each write data bit of a one shall not effect that bit.
 - zeroToToggle** means in a bitwise fashion each write data bit of a zero shall toggle the corresponding bit in the field, and each write data bit of a one shall not effect that bit.
 - iv) **writeValueConstraint** (optional) describes a set of constraint values that are the only values that can be written to this field. If **writeValueConstraint** is not present, no constraint values are defined for this field. See [6.14.11](#).
 - v) **readAction** (optional) describes an action that happens to a field after a read operation happens. **clear** indicates the field is cleared after a read operation. **set** indicates the field is

set after a read operation. **modify** indicates the field is modified in some way after a read operation. If **readAction** not specified, the field is not modified after a read operation. When set to the value **modify**, the **modify** attribute can be set to provide a user-defined enumeration providing additional information about the type of the modification.

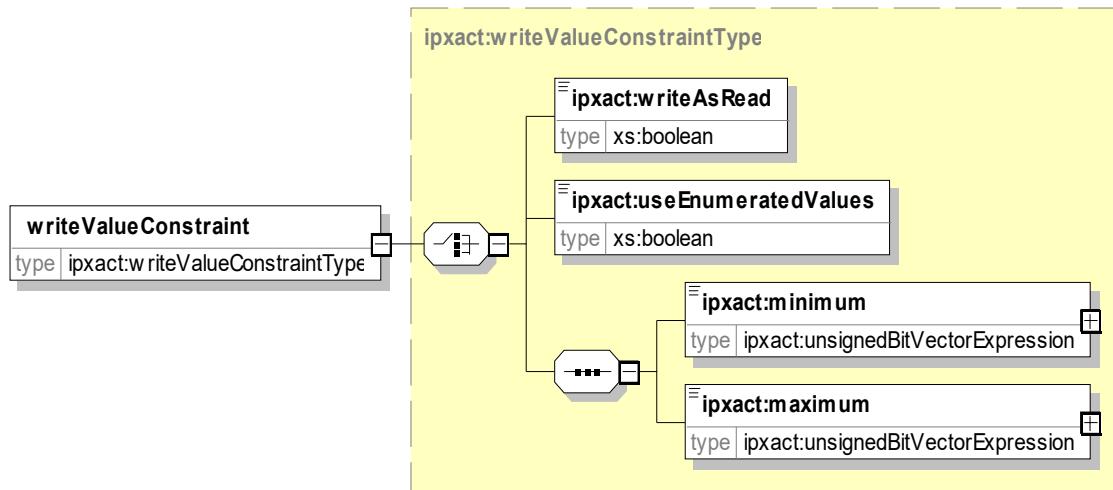
- vi) **readResponse** (optional) describes the value read after the read mux, while a **reset** value describes the value of field bits in reset. The expected read value of a field in a certain mode is described by the **readResponse** if that element is present. If that element is not present, then the **reset** value is expected to be read. Aliases do not need to have identical **readResponse** values.
 - 3) **broadcasts** (optional) contains one or more **broadCastTo** elements. Each **broadCastTo** element describes another field that is also written to by write operations to this field. Field broadcast does not contribute to volatility of fields because the broadcast is explicitly described in IP-XACT. The field is referenced using a **fieldReferenceGroup** describe in [C.13](#).
 - 4) **accessRestrictions** (optional) describes which bits of the field can be read and written for different operating modes. See [6.14.11](#).
 - 5) **testable** (optional; type: **boolean**) defines if the field is testable by a simple automated register test. If this is not present, testable is presumed to be **true**.
 - 6) **testConstraint** (optional; type: **string**; default: **unconstrained**) attribute defines the constraint for the field during a simple automated register test.
 - 7) **unconstrained** indicates there are no restrictions on the data that may be written or read from the field. **restore** indicates the field's value shall be restored to the original value before accessing another register. **writeAsRead** indicates the field shall be written only to a value just previously read from the field. **readOnly** indicates the field shall be only read.
 - 8) **reserved** (optional; type: **unsignedBitExpression**; default: **0**) indicates this field is reserved. No specific definition or interpretation of the meaning of reserved is made. See [C.7.9](#).
 - 9) **vendorExtensions** (optional) adds any extra vendor-specific data related to this field. See [C.27](#).
- b) **enumeratedValues** (optional) contains one or more **enumeratedValue** elements. The **enumeratedValue** element describes the name and values pairs for the given field. See [6.14.12](#).

See also [SCR 7.2](#), [SCR 7.4](#), [SCR 7.9](#), [SCR 7.10](#), [SCR 7.11](#), [SCR 7.12](#), [SCR 7.16](#), and [SCR 7.17](#).

6.14.10 Write value constraint

6.14.10.1 Schema

The following schema details the information contained in the **writeValueConstraint** element, which may appear as an element inside the **fieldAccessPolicy** element.



6.14.10.2 Description

The **writeValueConstraint** element describes a set of constraint values that are the only values that can be written to this field. If **writeValueConstraint** is not present, the legal values for this field are not defined. **writeValueConstraint** is one of the following:

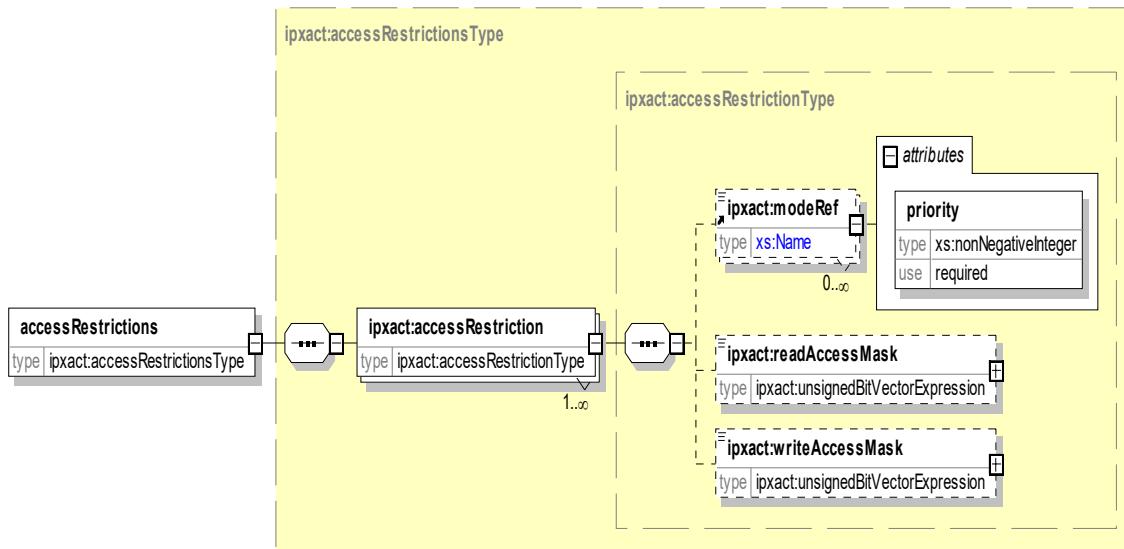
- writeAsRead** (type: *boolean*) if **true** implies the only legal value to write to this field is a value previously read from this field. If **writeAsRead** is not present, it is presumed to be **false**.
- useEnumeratedValues** (type: *boolean*) if **true** implies the only legal values to write to this field are the values specified in the **enumeratedValues** element for this field. If **useEnumeratedValues** is not present, it is presumed to be **false**.
- writeValueConstraint** can also be both of the **minimum** and **maximum** elements.
 - minimum** (type: *unsignedBitVectorExpression*, see [C.7.10](#)) contains the minimum value that may be written to this field.
 - maximum** (type: *unsignedBitVectorExpression*, see [C.7.10](#)) contains the maximum value that may be written to this field.

See also [SCR 7.10](#) and [SCR 7.11](#).

6.14.11 accessRestriction

6.14.11.1 Schema

The following schema details the information contained in the **accessRestrictions** element, which may appear as an element inside the **fieldAccessPolicy** element.



6.14.11.2 Description

The **accessRestriction** element describes which bits of the field can be read and written for different operating modes.

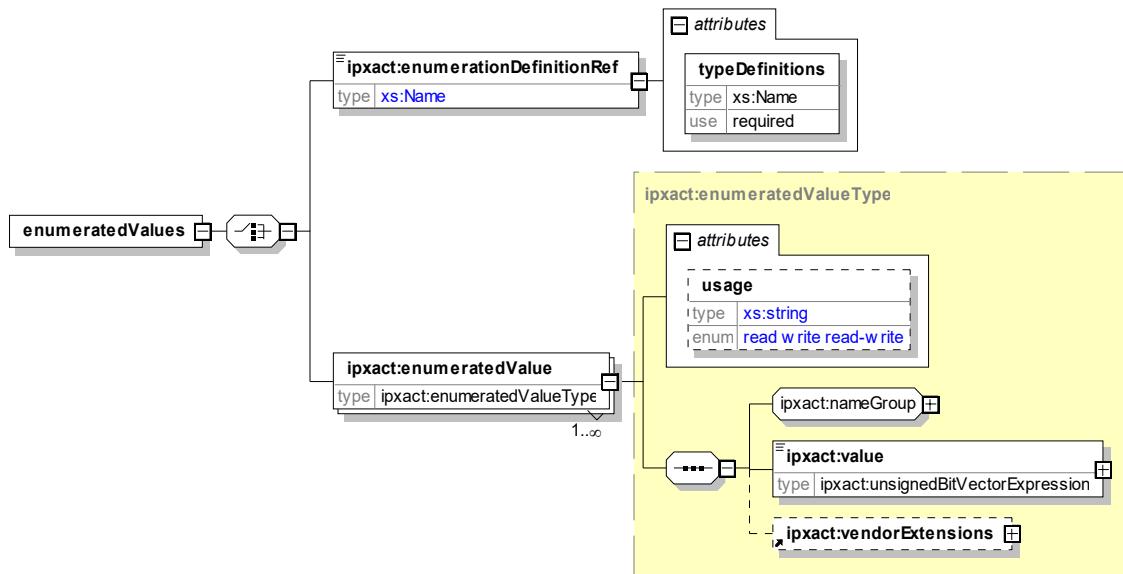
- modeRef** (optional) identifies the operating **mode** for which the **accessRestriction** element is active by referencing a mode name in the containing description and providing a priority. An **accessRestriction** is active if it references an active mode and that has highest priority among the active modes referenced in all **accessRestriction** elements of the **accessRestrictions** element. The **modeRef** element value shall be unique within the containing **accessRestrictions** element. The **modeRef** element has an attribute **priority** (mandatory; type *nonNegativeInteger*) which indicates the priority of the referenced mode for this **accessRestriction**. The lower the value, the higher the priority. The priority value of a **modeRef** element shall be unique in the scope of the referenced modes inside the **accessRestrictions** element. If no **modeRef** element is present, then the **accessRestriction** element applies to all modes.
- readAccessMask** (optional; type: *unsignedBitVectorExpression*) describes which bits of the field are readable in the referenced modes. For each bit in the field a '1' at the position in the mask means that the bit is readable and a '0' means that the bit is not readable. Reading non-readable bits will return undefined bits. See [C.3](#).
- writeAccessMask** (optional; type: *unsignedBitVectorExpression*) describes which bits of the field are writable in the referenced modes. For each bit in the field a '1' at the position in the mask means that the bit is writable and a '0' means that the bit is not writeable. Writing non-writeable bits will discard those bit values. See [C.3](#).

See also [SCR 7.30](#).

6.14.12 Enumeration values

6.14.12.1 Schema

The following schema details the information contained in the **enumeratedValues** element, which may appear as an element inside the **field** element.



6.14.12.2 Description

The **enumeratedValues** is a container element containing an **enumerationDefinitionRef** or an **enumeratedValue**. An **enumerationDefinitionRef** is a reference to an **enumerationDefinition** inside a **typeDefinitions** element. The reference is made through the name of an **externalTypeDefinitions** element. The **enumeratedValue** element describes the name and values pairs for the given field.

- usage** (optional; type: *string*; default: **read-write**) attribute defines the software access condition under which this name value pair is valid. Possible values are **read**, **write**, and **read-write**.
- nameGroup** group is defined in [C.19](#). All **name** elements shall be unique within the containing **enumeratedValues** element.
- value** (mandatory; type: *unsignedBitVectorExpression*, see [C.7.10](#)) defines the value to assign to the specified name.
- vendorExtensions** (optional) adds any extra vendor-specific data related to this enumeration. See [C.27](#).

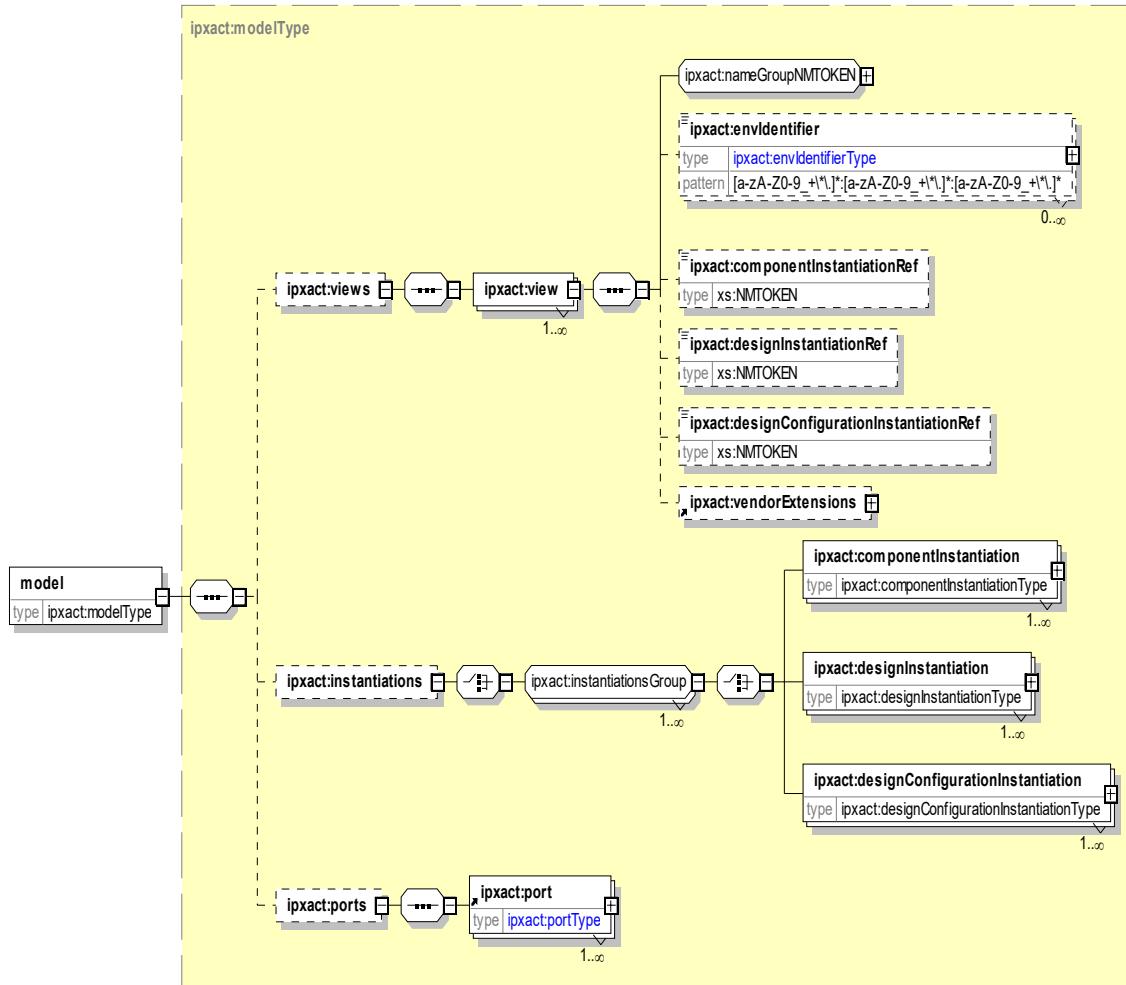
See also [SCR 7.10](#), [SCR 7.11](#), [SCR 7.18](#), and [SCR 7.19](#).

6.15 Models

6.15.1 Model

6.15.1.1 Schema

The following schema details the information contained in the **model** element, which may appear as an element inside the **component** element.



6.15.1.2 Description

The **model** element describes the views, instantiations, and ports of a component. A **model** element may contain the following:

- views** (optional) contains a list of all the views for this object. A component may have many different views. Each view can reference a component instantiation, a design instantiation, and a design configuration instantiation. A *component instantiation* may describe the source hardware module/entity with its pin interface. A *design instantiation* references an IP-XACT design representation. A *design configuration instantiation* configures an IP-XACT design configuration representation.

A **views** element describes an unbounded set of **view** elements. Each **view** element specifies a representation level of a component. It contains the following elements:

- 1) **nameGroupNMToken** group is detailed in [C.19.2](#). The **name** elements shall be unique within the containing **views** element.
- 2) **envIdentifier** (optional) designates and qualifies information about how this model view is deployed in a particular tool environment. The format of the element is a string with three fields separated by colons [:] in the format of *Language:Tool:VendorSpecific*. The regular expression that is used to check the string is `[A-Za-z0-9_+*\.]*:[A-Za-z0-9_+*\.]*:[A-Za-z0-9_+*\.]*` The fields are as follows:
 - i) *Language* indicates this view may be compatible with a particular tool, but only if that language is supported in that tool, e.g., different versions of some simulators may support two or more languages. In some cases, knowing the tool compatibility is not enough and may be further qualified by language compatibility, e.g., a compiled HDL model may work in a VHDL-enabled version of a simulator, but not in a SystemC-enabled version of the same simulator.
 - ii) *Tool* indicates this view contains information that is suitable for the named tool. This might be used if this view references data that is tool-specific and would not work generically, e.g., HDL models that use simulator-specific extensions.

Accellera may publish list of approved tool identification strings. These strings shall contain the tool name, as well as the company's domain name, separated by dots. Some examples of well-formed tool entries are

```
designcompiler.synopsys.com
ncsim.cadence.com
modelsim.mentor.com
```

This field can alternatively indicate generic tool family compatibility, including *Simulation and *Synthesis. To support transportability of created data files, it is recommended to use the published, generally recognized, tool designation, if available, when referencing a tool.
- iii) *VendorSpecific* can be used to further qualify tool and language compatibility. This can be used to indicate additional processing information may be required to use this model in a particular environment. For instance, if the model is a SWIFT simulation model, the appropriate simulator interface may need to be enabled and activated.

Any or all of the **envIdentifier** fields may be used. Where there are multiple environments to which a particular **view** is applicable, multiple **envIdentifier** elements can be listed.

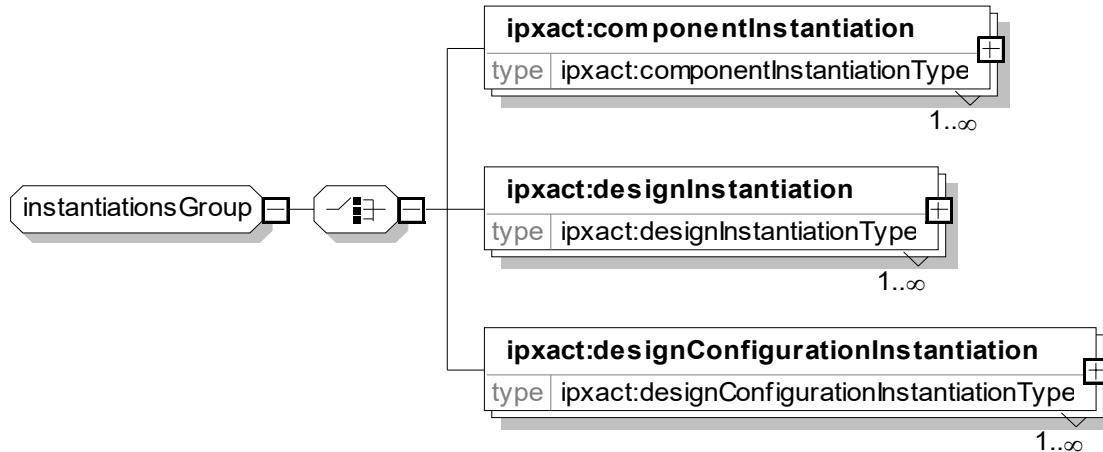
- 4) **componentInstantiationRef** (optional; type: *NMOKEN*) specifies a name that references a component inside the **instantiations** element.
- 5) **designInstantiationRef** (optional; type: *NMOKEN*) specifies a name that references a design inside the **instantiations** element.
- 6) **designConfigurationInstantiationRef** (optional; type: *NMOKEN*) specifies a name that references a design configuration inside the **instantiations** element.
- b) **instantiations** (optional) specifies the component, design, and design configuration instantiations for all views (as an **instantiationsGroup**). See [6.15.2](#).
- c) **ports** (optional) contains the list of ports for this object. A ports is an external connection from the object. See [6.15.7](#).

See also [SCR 1.16](#), [SCR 1.17](#), [SCR 5.8](#), and [SCR 6.26](#).

6.15.2 instantiationsGroup

6.15.2.1 Schema

The following schema details the information contained in the **instantiationsGroup** group, which is contained inside an **instantiations** element inside a **model** element.



6.15.2.2 Description

An **instantiationsGroup** group specifies the component, design, and design configuration instantiation for a particular view. There can be multiple occurrences of **instantiationsGroup** (see [6.15.1.1](#)), and each **instantiationsGroup** can have multiple elements from any branch. Each occurrence shall contain one or more of the following elements:

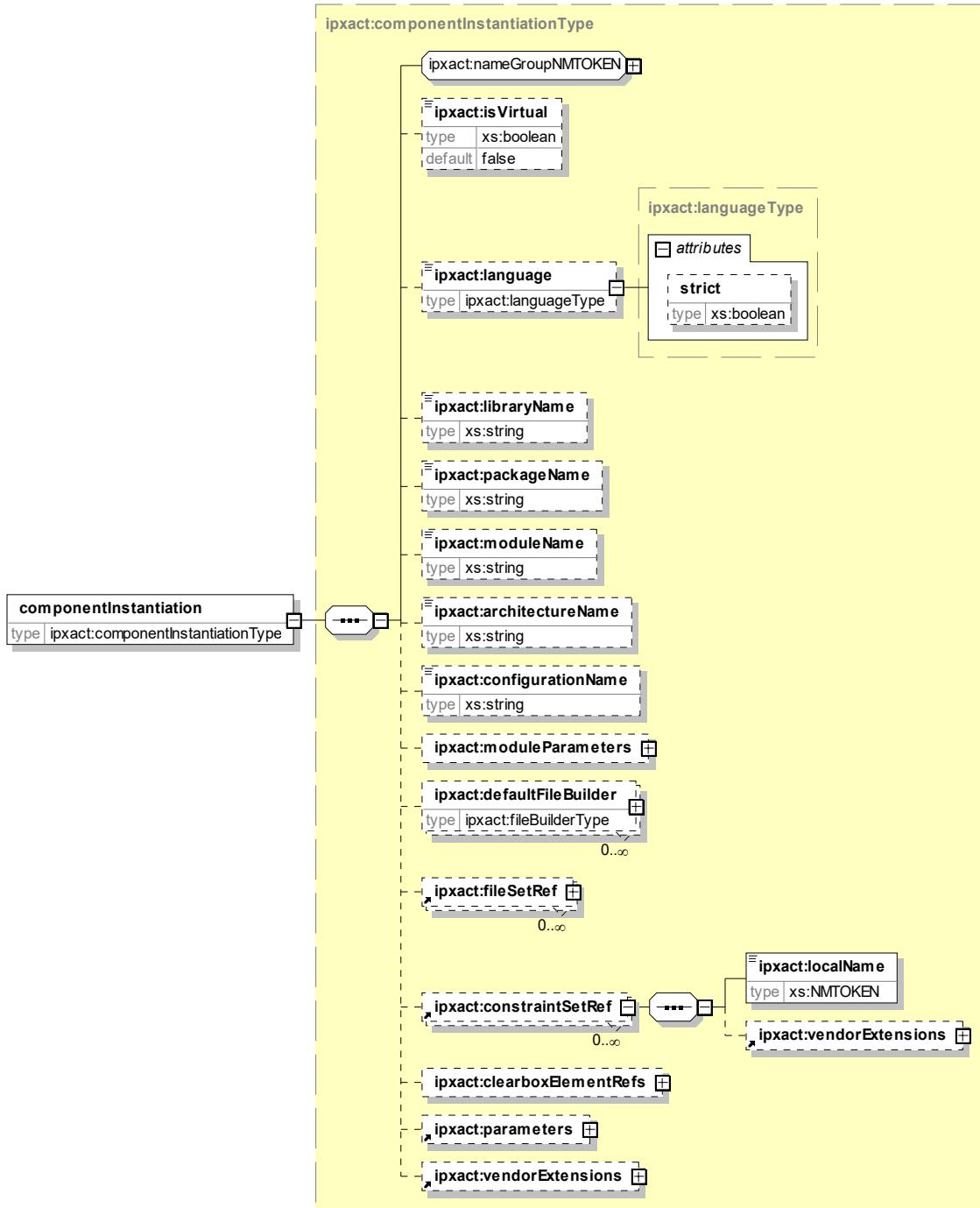
- componentInstantiation** (type: *componentInstantiationType*) specifies information concerning the instantiation of this component. See [6.15.3](#).
- designInstantiation** (type: *designInstantiationType*) specifies information concerning the internal (design) connectivity of this component, including a reference to an IP-XACT **design** document. See [6.15.4](#).
- designConfigurationInstantiation** (type: *designConfigurationInstantiationType*) specifies information concerning the internal (design) configuration of this component, including a reference to an IP-XACT **designConfiguration** document defining the configured hierarchy of this component. See [6.15.5](#).

See also [SCR 5.14](#), [SCR 5.15](#), and [SCR 5.23](#).

6.15.3 componentInstantiation

6.15.3.1 Schema

The following schema details the information contained in the **componentInstantiation** element, which is the type of the component element within a **model/instantiations** element.



6.15.3.2 Description

An **componentInstantiation** element details how a specific component instance is associated with a view. It has the following elements:

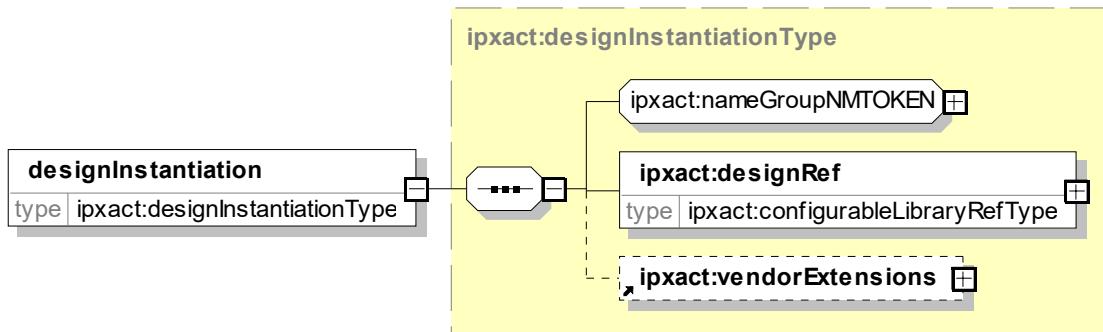
- a) **nameGroupNMTOKEN** group is detailed in [C.19.2](#). The **name** elements shall be unique within the containing **instantiations** element.
- b) **isVirtual** (optional; type: **boolean**; default: **false**) when **true**, indicates this component shall not be netlisted.
- c) **language** (optional) specifies the HDL used for a specific instantiation, for example, verilog or vhdl. The **language** element is of type **languageType**, which is a token string along with an optional boolean attribute **strict** (type: **boolean**; default: **false**). If **true**, the language shall be strictly enforced. If this attribute is not present, its effective value is **false**.
- d) **libraryName** (optional) is a string that specifies the library name in which the model shall be compiled. If the **libraryName** element is not present, then its value defaults to **work**.
- e) **packageName** (optional; type: **string**) is a string that describes the VHDL package containing the interface of the model. If the **packageName** element is not present, there is no VHDL package.
- f) **moduleName** (optional; type: **string**) is a string that describes the Verilog, SystemVerilog, or SystemC module name or the VHDL entity name. If the **moduleName** element is not present, then its value defaults to the component VLVN **name**.
- g) **architectureName** (optional; type: **string**) is a string that describes the VHDL architecture name. If the element is not present then there is no VHDL architecture.
- h) **configurationName** (optional; type: **string**) is a string that describes the Verilog, SystemVerilog, or VHDL configuration name. If the element is not present, then there is no VHDL configuration.
- i) **moduleParameters** (optional) specifies a parameter name-value pair container. See [6.15.6](#).
- j) **defaultFileBuilder** (optional) is an unbounded list of default file builder options for the **fileSets** referenced in this **componentInstantiation**. This contains all the same elements as the element **fileBuilder**. See [6.17.4](#).
- k) **fileSetRef** (optional) is an unbounded list of references to **fileSet** names within the containing document. See [C.15](#).
- l) **constraintSetRef** (optional) is an unbounded list of references to constraint sets, valid timing constraints for a view. **constraintsSets** are defined only for **wire** style **ports**. The **constraintSetRef** element is of type **NMTOKEN**. See [6.15.14](#).
- m) **clearboxElementRefs** (optional) contains references to clear box elements of a component that are valid for this view. If the view contains an implementation of any of the clear box elements for the component, the **view** section shall include a reference to that **clearbox** element, with a string providing a language-dependent path to enable the DE to access the **clearbox** element. See [6.19](#).
- n) **parameters** (optional) details any additional parameters that describe the instantiation. See [C.21](#).
- o) **vendorExtensions** (optional) adds any extra vendor-specific data related to the instantiation. See [C.27](#).

See also [SCR 6.27](#).

6.15.4 designInstantiation

6.15.4.1 Schema

The following schema details the information contained in the ***designInstantiation*** element, which is the type of the design element within a ***model/instantiations*** element.



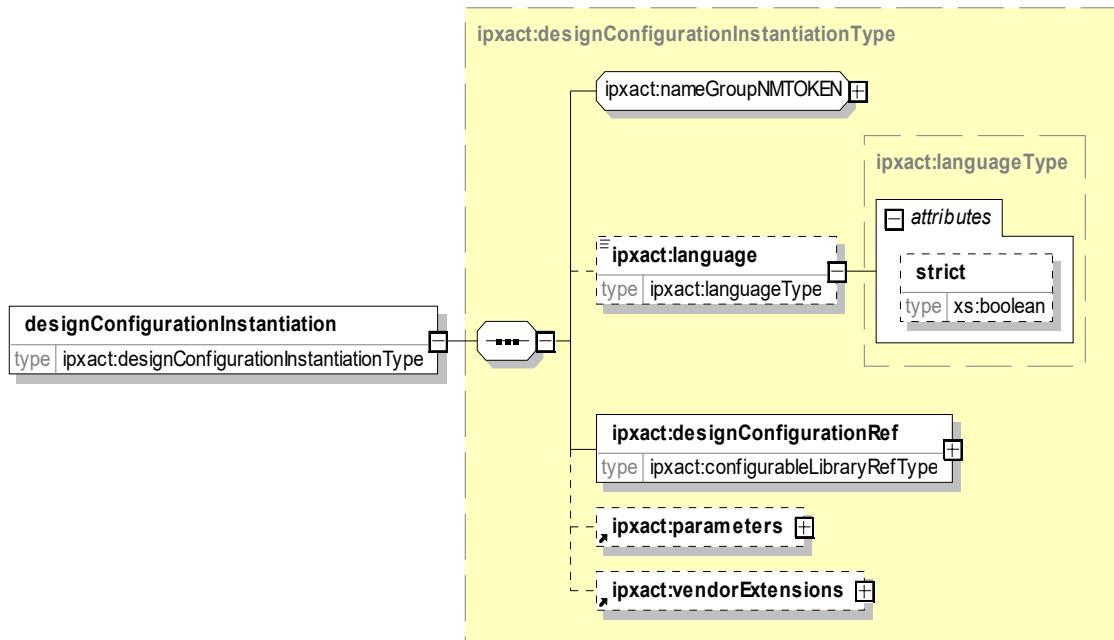
A ***designInstantiation*** element details how a specific design instance is associated with a view. It has the following elements:

- a) ***nameGroupNMTOKEN*** group is detailed in [C.19.2](#). The ***name*** elements shall be unique within the containing ***instantiations*** element.
- b) ***designRef*** (mandatory; type: ***configurableLibraryRefType***, see [C.10](#)) specifies the design description and configuration values to applied to the design description.
- c) ***vendorExtensions*** (optional) adds any extra vendor-specific data related to the view. See [C.27](#).

6.15.5 designConfigurationInstantiation

6.15.5.1 Schema

The following schema details the information contained in the **designConfigurationInstantiation** element definition, which is the type of the design configuration element within a **model/instantiations** element.



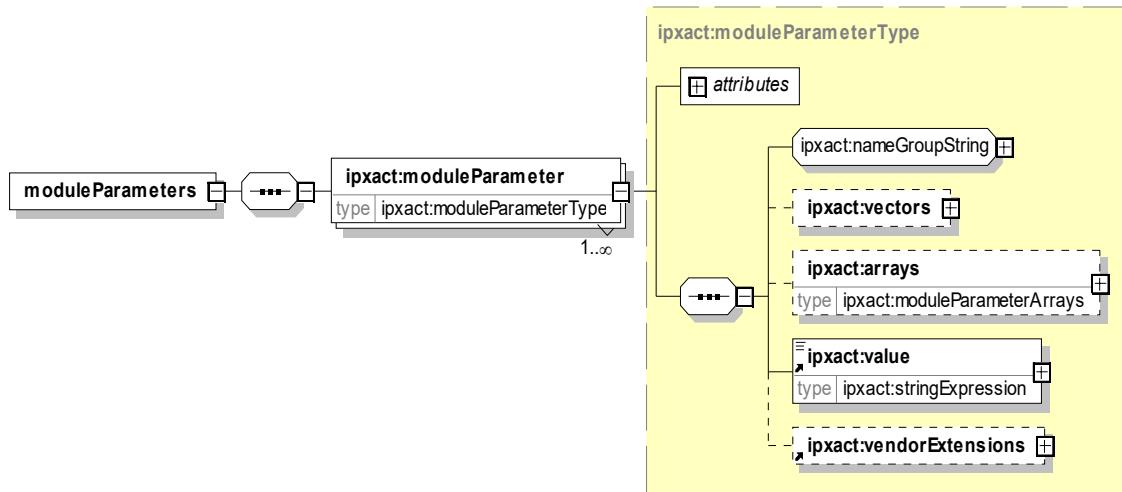
A **designConfigurationInstantiation** element details how a specific design configuration instance is associated with a view. It has the following elements:

- nameGroupNMToken** group is detailed in [C.19.2](#). The **name** elements shall be unique within the containing **views** element.
- language** (optional) specifies the HDL used for a specific instantiation, for example, verilog or vhdl. The **language** element is of type **languageType**, which is a token string along with an optional boolean attribute **strict** (type: **boolean**; default: **false**). If **true**, the language shall be strictly enforced. If this attribute is not present, its effective value is **false**.
- designConfigurationRef** (mandatory; type: **configurableLibraryRefType**, see [C.10](#)) specifies the design configuration description.
- parameters** (optional) details any additional parameters that describe the instantiation. See [C.21](#).
- vendorExtensions** (optional) adds any extra vendor-specific data related to the instantiation. See [C.27](#).

6.15.6 Module parameters

6.15.6.1 Schema

The following schema details the information contained in the **moduleParameters** element, which may appear as an element inside a **component/model/instantiations/componentInstantiation** element.



6.15.6.2 Description

A **moduleParameters** element contains an unbounded list of **moduleParameter** elements. The **moduleParameter** element describes a name-value pair used to configure the model of this component instance. A **moduleParameter** contains the following elements and attributes:

- a) Attributes used to constrain the parameter value and enable presentation to the user as defined in [C.6. moduleParameter](#) also has the following attributes:
 - 1) **dataType** (optional; type: *string*) attribute specifies the data type as it pertains to the language of the model. This definition is used to define the type for component declaration and as such has no semantic meaning. For example, SystemC could be `int`, `double`, `char*`, etc. For VHDL, this could be `std_logic`, `std_logic_vector`, `integer`, etc.
 - 2) **usageType** (optional; default: *nontyped*) attribute specifies how this parameter is used in different modeling languages: **nontyped**, **typed**, and **runtime**. See [6.15.6.2.1](#).
 - 3) **dataTypeDefinition** (optional; type: *string*) attribute specifies the file containing the definition of the **dataType** definition. This file could be referenced in a **filesetRef** itself referenced by the **componentInstantiation**.
 - 4) **constrained** (optional; type: *list of strings*) attribute indicates for which vectors and arrays the number of bits in the type declaration is fixed by referencing the **vectorId** and **arrayId** values. The **vectorId** and **arrayId** values are separated by white space. If the number of bits is fixed (**constrained** is **true**), the bit indices are not required when referencing the type. When **constrained** is not present, bit indices are required on all references to the module parameter.
 - b) **nameGroupString** group is defined in [C.19.5](#). For **moduleParameter**, the name element shall be unique within the containing **componentInstantiation** element.
 - c) **vectors** (optional) specifies dimensions required to define a vectored value type (see [C.26.2](#)).
 - d) **arrays** (optional; type: **moduleParameterArrays**, see [C.4.1](#)) specifies dimensions required to define an arrayed value type.

- e) **value** (mandatory; type: *complexBaseExpression*, see [C.7](#)) contains the value of this module parameter. This needs to meet the type requirements defined within the attributes on this element (see [C.6](#)).
- f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See [C.27](#).

See also [SCR 5.2](#).

6.15.6.2.1 Typed, non-typed, and runtime parameters classification

There are three categories of parameters: typed, non-typed, and runtime.

The *typed* parameters (or declaration parameters) appear in object-oriented (OO) languages such as C++/SystemC or SystemVerilog. A 'typed' parameter is a parameter of the type in the target domain. Parameters of a type and therefore 'typed parameters' are 'generics' (VHDL) or 'parameters' in Verilog, SystemVerilog or 'Class template parameters' in C++/SystemC.

In C++/SystemC, these are named `Class` template parameters. Templates can be used to develop a generic class prototype (specification), which can be instantiated with different data types. This is very useful when the same kind of class is used with different data types for individual members of the class. Parameterized types are used as data types, and then a class can be instantiated, i.e., constructed and used by providing arguments for the parameters of the class template. A class template is a specification of how a class should be built (i.e., instantiated) given the data type or values of its parameters.

Class template parameters can have default arguments, which are used during class template instantiation when arguments are not provided. Because the provided arguments are used starting from the far left parameter, default arguments should be provided for the right-most parameters.

Example 1

```
template <typename T>
class FIFO {
    FIFO();
    T pull();
    void push(T &x);
};
```

In SystemVerilog, typed parameters are named type parameters. Type parameters can be used in SystemVerilog classes, interfaces, or modules to provide the basic function of C++ templates.

Example 2

```
typedef bit[32] DataT;
interface FIFO #(type T);
    Method T pull();
    Method push (T x);
endinterface: FIFO
```

A 'nontyped' parameter refers to an argument in a specific function in the target domain. Non-typed parameters are arguments in that function signature. The agreement is usually that the 'specific' function is the constructor of the type in the target domain. That means for non-OO target domains there is no mapping of 'non-typed' parameters.

These two kinds of parameters (typed and non-typed) can be combined to model complex IP modules.

Example 4

In SystemC:

```
template <typename T> // type parameter
class testModule : public sc_module {
public:
    testModule(sc_module_name modnamemodname, string
    portname) :
        // non type parameters
        sc_module(modname),
        testport(portname) {...}
        sc_port<T> testport;
};
```

In a top SystemC netlist design, such a class is instantiated as follows:

```
testModule<bool> test("myModuleName", "port1");
```

The *runtime* parameter is a parameter whose value is set at run-time.

[Table 4](#) illustrates the various combinations of target domain and different parameter kinds.

Table 4—Target domain combinations and parameter mappings

Target domain	Parameter kind ‘type’ (elaboration time constants)	Parameter kind, ‘non- typed,’ (runtime constants in the sense that the value supplied to the invoked function needs only be present right before the function is invoked)	Parameter kind ‘runtime’
VHDL	VHDL generic	NA	NA
Verilog	Module parameter	NA	Plusargs (+top.foo.bla.MYPARAM=5)
SystemVerilog	Template parameter mymoduletype#(a+5) myinst()	Constructor arg myinst foo() = new(a+5)	Plusargs (+top.foo.bla.MYPARAM=, A+5' ?? ='A\${ENV-VAR}' ??)
SystemC	Template parameter	Constructor argument	SystemC CCI
UVM-SystemVerilog	Template parameter	The ctor is not freely available here. alternate mechanisms would be to use an alternate function or uvm_config_db)	UVM specific plus args (+uvm_set_config...)
UVM-SystemC	Template parameter	Constructor argument	SystemC CCI

6.15.6.2.2 Generic parameters mapping in different languages

[Table 5](#) summarizes the two kinds of parameters (initialization and declaration) expressed in the four most commonly used hardware languages.

A *declaration parameter* (e.g., `int`) shall be used when declaring an IP instance in a top netlist (e.g., `myIP int myIntIP;`). An *initialization parameter* (e.g., `myName`) shall be used when initializing the instance of that IP (e.g., `myIntIP("myName");`).

Backward compatibility affecting items marked with an asterisk (*).

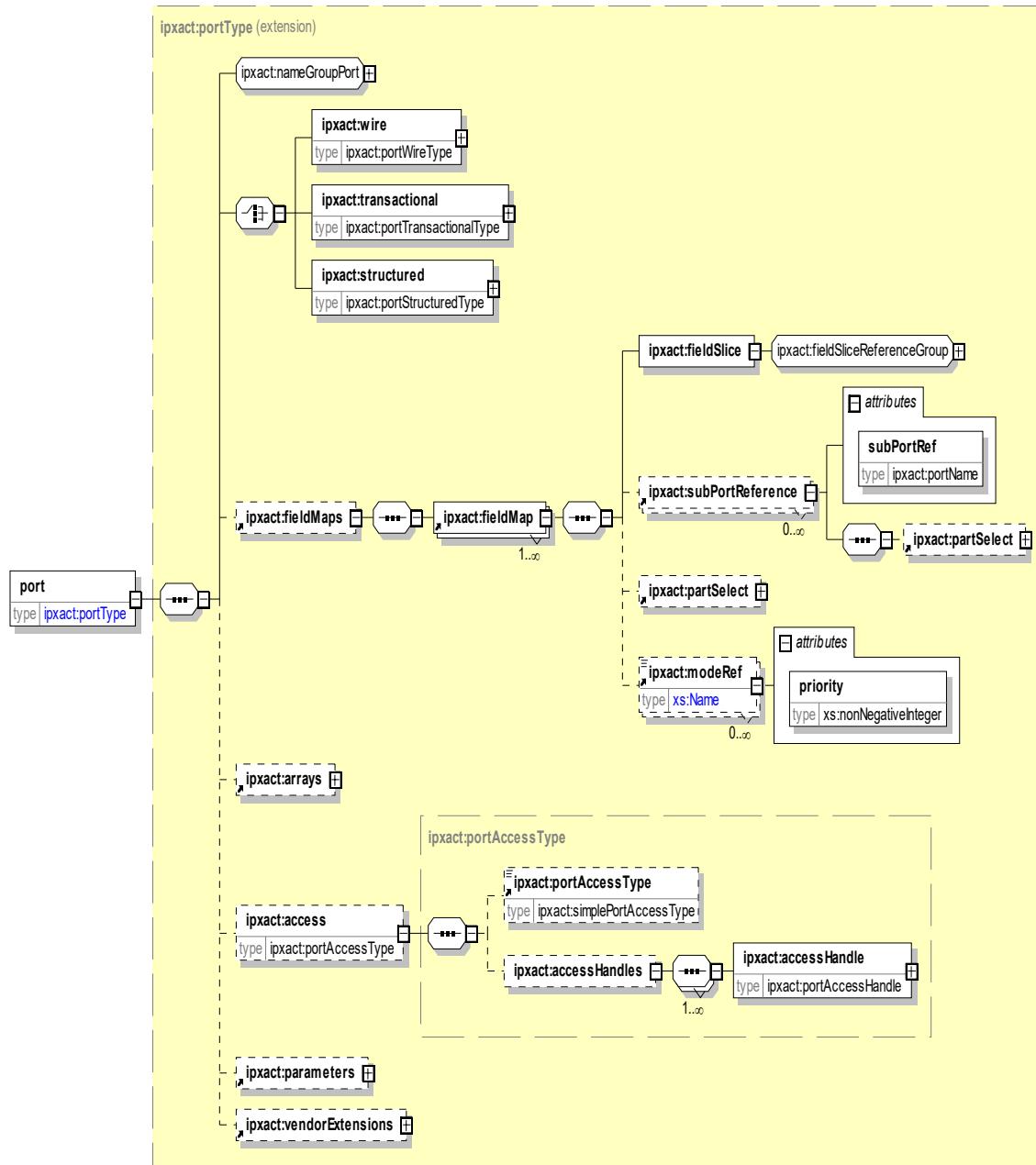
Table 5—Parameter mappings

Target domain	non-typed	typed
VHDL	NA	Generics*
Verilog	NA	(module) Parameter*
SystemVerilog	Constructor*	(class/module) Parameter
SystemC	Constructor	Class template parameter
UVM-SystemVerilog	The ctor is not freely available here. alternate mechanisms would be to use an alternate function or <code>uvm_config_db</code>	(class/module) parameter
UVM-SystemC	Constructor	Class template parameter

6.15.7 Component ports

6.15.7.1 Schema

The following schema defines the information contained in the **ports** element, which may appear within a **component**.



6.15.7.2 Description

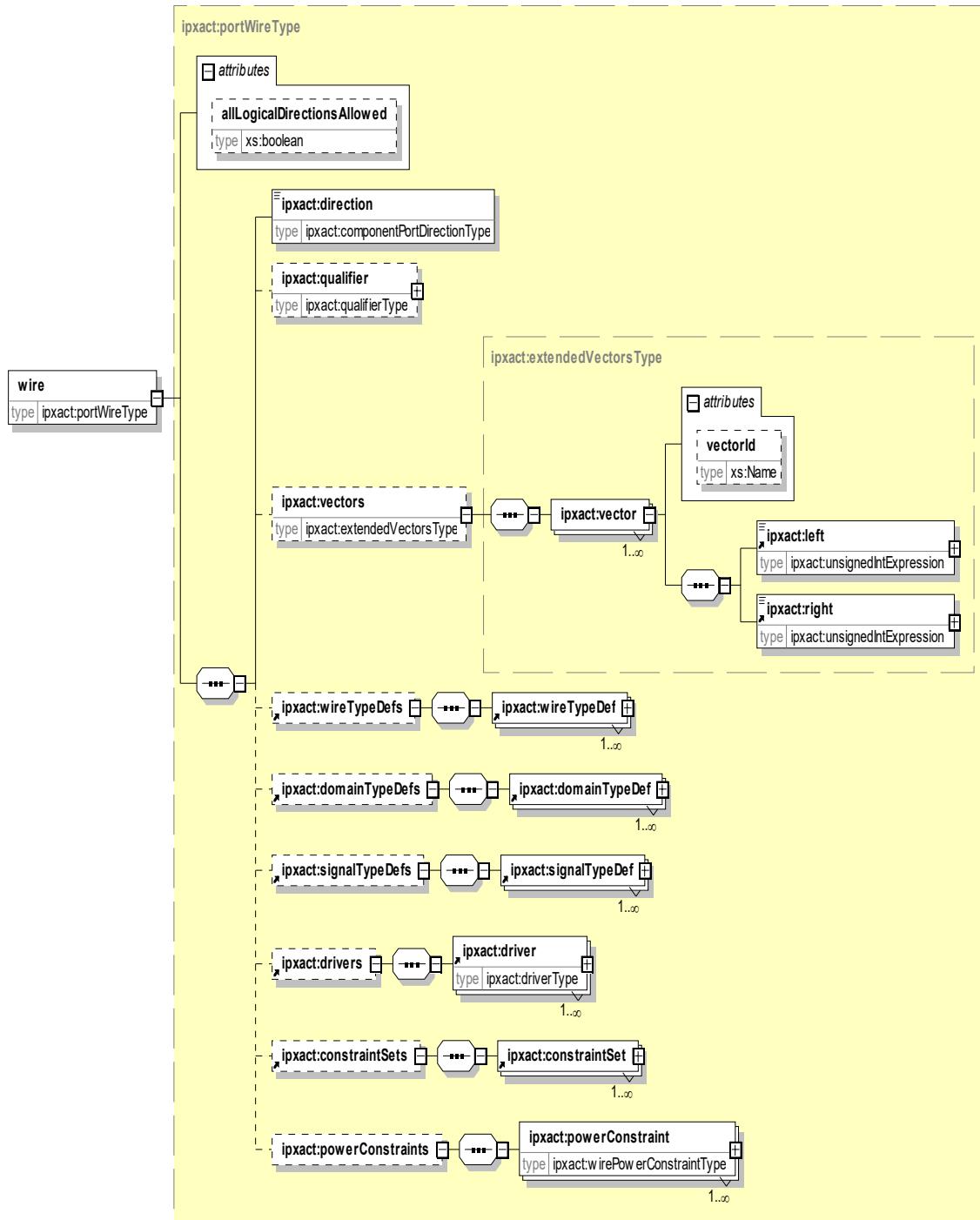
The **ports** element defines an unbounded list of **port** elements. Each **port** element describes a single external port on the component.

- a) **nameGroupPort** group is defined in [C.19.4](#). The **name** elements shall be unique within the containing **ports** element.
- b) Each **port** shall be described as a **wire**, **transactional**, or a **structured** port.
 - 1) **wire** defines ports that transport purely binary values. Values may be aggregated using vectors and/or arrays. See [6.15.8](#).
 - 2) **transactional** defines all other styles of ports, typically used for TLM. See [6.15.19](#).
 - 3) **structured** defines a port that is a structure, union, or SystemVerilog interface composed out of binary values. See [6.15.23](#).
- c) **fieldMaps** (optional) contains one or more **fieldMap** elements. Each **fieldMap** element describes a mapping of register field bits onto port bits indicating that the register field bits are accessible through the port bits for given operating modes. Each **fieldMap** has the following elements:
 - 1) **fieldSlice** (mandatory) describing register field bits using a **fieldSliceReferenceGroup** See [C.14](#).
 - 2) **subPortReference** (optional) describing a segment in a structured port if the encapsulating port is structured. The attribute **subPortRef** references a **subPort** of the encapsulating structured port. The **partSelect** element describes a slice of the referenced **subPort**.
 - 3) **partSelect** (optional) describes a slice of the referenced **structured** port.
 - 4) **modeRef** (optional) references a mode of the encapsulating component in which this **fieldMap** is active. If no **modeRef** is present, then the **fieldMap** is always active. A **fieldMap** is active if it references an active mode and it has highest priority among the active modes referenced in all **fieldMap** elements of the **fieldMaps** element. The **modeRef** element value shall be unique within the containing **fieldMap** element. The **modeRef** element has an attribute **priority** (mandatory; type *nonNegativeInteger*) which indicates the priority of the referenced mode for this **fieldMap**. The lower the value, the higher the priority. The priority value of a **modeRef** element shall be unique in the scope of the referenced modes inside the **fieldMap** element.
- d) **arrays** (optional type: *configurableArrays*, see [C.4.1](#)) specifies dimensions for a port defined as an array.
- e) **access** (optional) defines the access for a port.
 - 1) **portAccessType** (optional; default: **ref**) indicates to a netlister how to access the port. The **portAccessType** has one of two possible values **ref** or **ptr**. If **ref**, a netlister should access the port directly (i.e., by reference), and if **ptr**, it should access the port with a pointer.
 - 2) **accessHandles** (optional) indicates to a netlister the method to be used to access the object representing the port when references need to be done using something other than the simple port name. This is typically a function call. The **accessHandles** element contains a list of **accessHandle** elements of type **portAccessHandle**. See [C.1.3](#).
- f) **parameters** (optional) specifies any parameter value(s) for this port. See [C.21](#).
- g) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See [C.27](#).

6.15.8 Component wire port Types

6.15.8.1 Schema

The following schema details the information contained in the **wire** element, which may appear as an element inside the top-level **component/model/port** element.



6.15.8.2 Description

The **wire** element describes the properties for ports that are of a wire style. A port can come in three different styles, wire, structured, or transactional. A wire port can have four different signal representations, continuous-conservative, continuous-non-conservative, discrete, and digital. A digital wire port applies for all scalar types (e.g., VHDL `std_logic` and Verilog `wire`) and vectors of scalars. Scalar types in VHDL also include integer and enumeration values; however, scalars in IP-XACT include only binary values that relate to a single wire in a hardware implementation.

A digital wire port transports purely binary values or vectors of binary values; it does not support tri-state or multiple strength values.

Continuous-conservative, continuous-non-conservative, and discrete wire port are explained later; see [6.15.10.2](#).

The **wire** element contains the following attributes and elements:

- a) **allLogicalDirectionsAllowed** (optional; type: `boolean`; default: `false`) attribute defines whether the port may be mapped to a port in an **abstractionDefinition** with a different direction. See [5.3](#).
- b) **direction** (mandatory) specifies the direction of this port: **in** for input ports, **out** for output ports, and **inout** for bidirectional and tri-state ports. **phantom** can also be used to define a port that exists only on the IP-XACT component, but not on the implementation referenced from the view.
- c) **qualifier** (optional) indicates which type of information this wire port carries. See [5.6](#).
- d) **vectors** (optional) specifies the dimensions for a non-scalar port. See [C.26.2](#).
- e) **domainTypeDefs** (optional) describes the ports domain as defined by the implementation. See [6.15.9](#).
- f) **signalTypeDefs** (optional) describes the ports signal representation as defined by the implementation. See also [6.15.10](#).
- g) **wireTypeDefs** (optional) describes the ports type as defined by the implementation. See [6.15.11](#).
- h) **drivers** (optional) defines an unbounded list of driver elements, defining drivers that may be attached to this port if no other object is connected to this port. This allows the IP to define the default state of unconnected inputs. A wire style port may define a **driver** element for a port only if the direction of the port is **in** or **inout**. See also [6.15.12](#).

If multiple drivers are specified, the **range** element shall be used to indicate which bits are driven by which driver, and no overlap bit ranges may be specified.

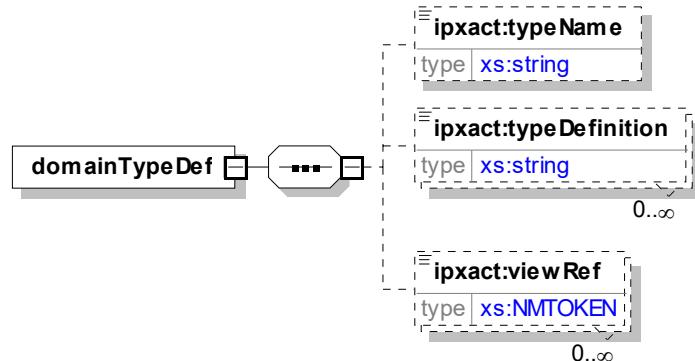
- i) **constraintSets** (optional) defines multiple set of constraints on a port used for synthesis or other operations. See [6.15.16](#).
- j) **powerConstraints** (optional) describes the ports power domain. See [C.24.2](#).

See also [SCR 6.5](#), [SCR 6.6](#), [SCR 6.7](#), and [SCR 6.12](#).

6.15.9 domainTypeDefs

6.15.9.1 Schema

The following schema details the information contained in the **domainTypeDefs** element.



6.15.9.2 Description

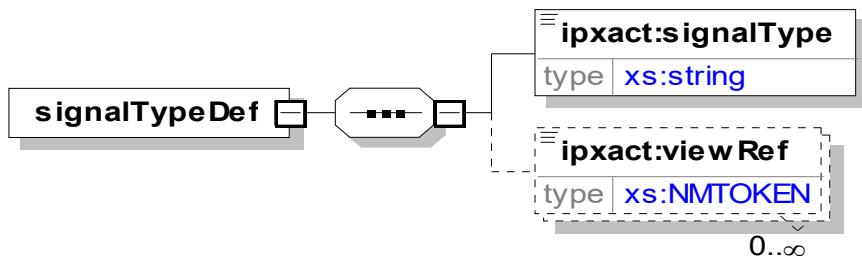
The **domainTypeDef** element describes disciplines of the encapsulating port. The **domainTypeDef** element definition contains the following elements:

- typeName** (mandatory; type: *string*) defines the name of the domain type for the port. For VHDL, some typical values would be discrete and electrical.
- typeDefinition** (optional; type: *string*) contains a language-specific reference to where the given type is actually defined. There can be multiple **typeDefinitions** for each port.
- viewRef** (optional) indicates the view or views in which this type definition applies. Multiple views can use the same set of type properties by specifying multiple **viewRef** elements. The **viewRef** shall match a view name in the containing object. See [C.29](#).

6.15.10 signalTypeDefs

6.15.10.1 Schema

The following schema details the information contained in the **signalTypeDefs** element.



6.15.10.2 Description

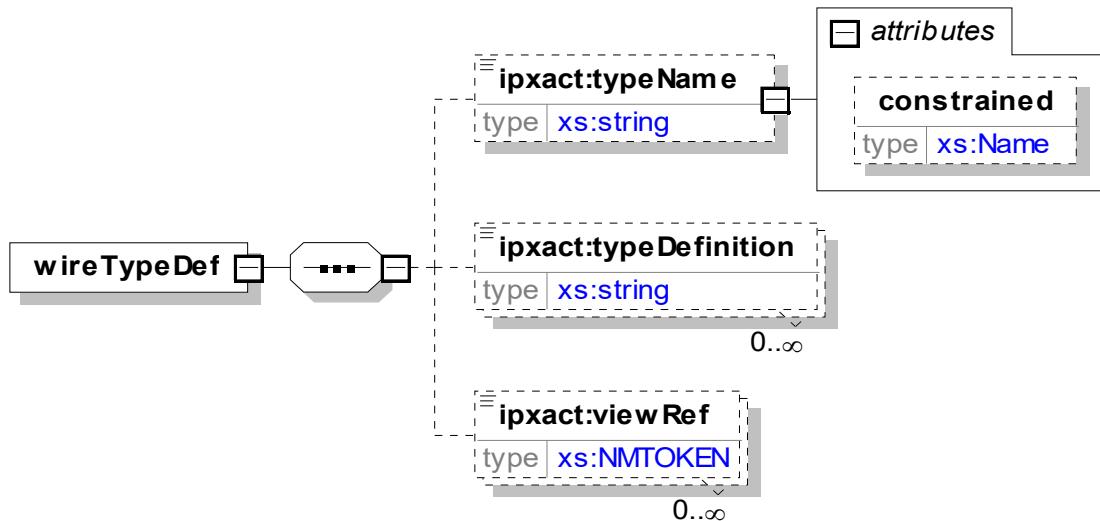
The **signalTypeDef** element describes signal representations of the encapsulating port. The **signalTypeDef** element definition contains the following elements:

- a) **signalType** (mandatory) defines the signal type for the port. **Continuous-conservative** is a signal that is continuous in time and quantity, where the quantity represents voltage and current to capture the conservative behavior of an electrical network (i. e., Kirchhoff's Laws apply), and that has no direction. **Continuous-non-conservative** is a directed signal which is continuous in time and quantity, where the quantity represents either a voltage or a current (often called signal flow). In this case it only captures the non-conservative behavior of a set of interconnected components. **Discrete** is a directed sampled signal which is time-discrete and value-continuous (the value may also be quantified), where the value represents either a voltage or a current (other representations are possible also). It captures the non-conservative behavior of a set of interconnected components. **Digital** is a time-discrete and value-discrete signal. This is the default value if **signalType** is not present.
- b) **viewRef** (optional) indicates the view or views in which this type definition applies. Multiple views can use the same set of type properties by specifying multiple **viewRef** elements. The **viewRef** shall match a **view name** in the containing object. See [C.29](#).

6.15.11 Component wireTypeDef

6.15.11.1 Schema

The following schema details the information contained in the **wireTypeDef** element, which may appear as an element inside the **wire** element of a top-level wire style **port** element. These elements define the definition type name, where the type is defined, and which views of a component or an abstractor use this type.



6.15.11.2 Description

The **wireTypeDef** element describes the type properties for a port per view of a component or abstractor. There can be an unbounded series of **wireTypeDefs** defined for each port, allowing the type properties to be defined differently for each view. If a port does not have **wireTypeDefs**, the default **wireTypeDef** applies for each view (see [Table 7](#)).

wireTypeDef contains the following elements:

- a) **typeName** (optional; type: *string*) defines the name of the type for the port. For VHDL, some typical values would be `std_logic` and `std_ulogic`.
 - 1) **constrained** (optional; type: *list of strings*) attribute indicates for which vectors and arrays the number of bits in the type declaration is fixed by referencing the `vectorId` and `arrayId` values. The `vectorId` and `arrayId` values are separated by white space. If the number of bits is fixed (i.e., `vectorId` or `arrayId` appear in the **constrained** list), the bit indices are not required when referencing the type. When **constrained** is not present, bit indices are required on all references to the type definition. See [6.15.11.2.1](#).
- b) **typeDefinition** (optional; type: *string*) contains a language-specific reference to where the given type is actually defined. [Table 6](#) shows some examples. There can be multiple **typeDefinitions** for each port.
- c) **viewRef** (optional; type: *NMOKEN*) specifies the views to which the **wireTypeDef** applies. See [C.29](#).

Also see [SCR 6.51](#).

Table 6—typeName and typeDefinition examples

Language	Meaning	typeName	typeDefinition
VHDL	A “Use” statement text. For example: USE IEEE.std_logic_1164.all	<code>std_logic</code>	<code>IEEE.std_logic_1164.all</code>
Verilog	Does not apply for Verilog.		
SystemC	A type defined externally (in a specific namespace). For example: <code>myNameSpace:::myType</code> defined in file <code>file.h</code> .	<code>myNameSpace:::myType</code>	<code>file.h</code>
SystemVerilog	A typedef declared in a package. For example: <code>myPackage:::mytype</code> defined in file <code>file.sv</code> .	<code>myPackage:::myType</code>	<code>file.sv</code>

6.15.11.2.1 Constrained and unconstrained array types

A *constrained array type* is a type for which the indices of the array have been specified in the definition. An *unconstrained array type* is a type for which the indices of the array have not been specified in the definition. An example of the RTL and corresponding IP-XACT is shown below.

```

type BYTE is array (7 downto 0) of std_logic; -- constrained
type my_logic_vector is array (NATURAL RANGE <>) of std_logic; -- unconstrained
entity example is
  port(
    A: out BYTE;
    B: out my_logic_vector(7 downto 0)
  );
end example;

<ipxact:port>
  <ipxact:name>A</ipxact:name>

```

```

<ipxact:wire>
  <ipxact:direction>out<ipxact:direction/>
  <ipxact:vectors>
    <ipxact:vector vectorId="vid">
      <ipxact:left>7</ipxact:left>
      <ipxact:right>0</ipxact:right>
    </ipxact:vector>
  </ipxact:vectors>
  <ipxact:wireTypeDefs>
    <ipxact:wireTypeDef>
      <ipxact:typeName ipxact:constrained="vid">BYTE</ipxact:typeName>
      <ipxact:typeDefinition>MYLIB.MYPKG.all</ipxact:typeDefinition>
      <ipxact:viewRef>VHDLsimView</ipxact:viewRef>
    </ipxact:wireTypeDef>
  </ipxact:wireTypeDefs>
  </ipxact:wire>
</ipxact:port>
<ipxact:port>
  <ipxact:name>B</ipxact:name>
  <ipxact:wire>
    <ipxact:direction>out<ipxact:direction/>
    <ipxact:vectors>
      <ipxact:vector>
        <ipxact:left>7</ipxact:left>
        <ipxact:right>0</ipxact:right>
      </ipxact:vector>
    </ipxact:vectors>
    <ipxact:wireTypeDefs>
      <ipxact:wireTypeDef>
        <ipxact:typeName>my_logic_vector</ipxact:typeName>
      <ipxact:typeDefinition>MYLIB.MYPKG.all</ipxact:typeDefinition>
      <ipxact:viewRef>VHDLsimView</ipxact:viewRef>
    </ipxact:wireTypeDef>
  </ipxact:wireTypeDefs>
  </ipxact:wire>
</ipxact:port>

```

6.15.11.2.2 Defaults

wireTypeDefs do not need to be defined for every view of a port. IP-XACT provides for these defaults based on the language of the view, as shown in [Table 7](#). For languages not shown here, no defaults can be presumed.

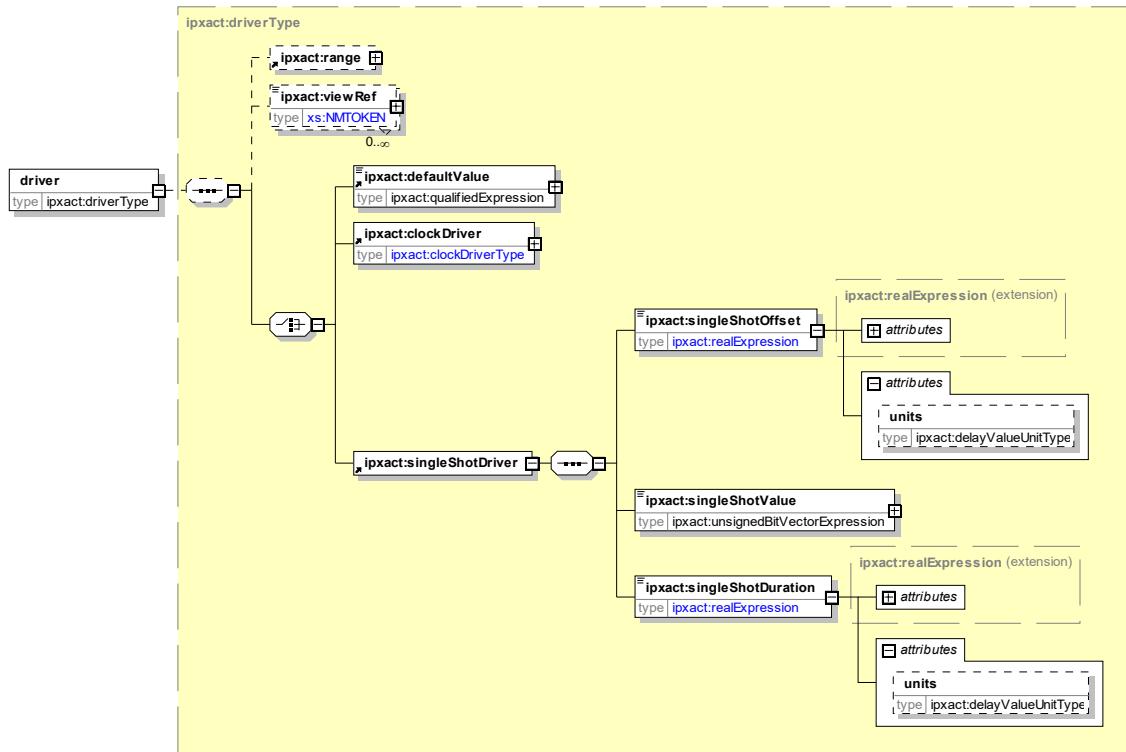
Table 7—View defaults

Language	Single bit	Vectors
VHDL	std_logic	std_logic_vector
Verilog	wire	wire
SystemC	sc_logic	sc_lv
SystemVerilog	logic	logic

6.15.12 Component driver

6.15.12.1 Schema

The following schema details the information contained in the **driver** element, which may appear as an element inside the **wire/driver** element of a top-level wire style **port** element. This element defines the type and value(s) to drive on this port when it is not connected in a design; it is allowed only on ports with the direction **in** or **inout**.



6.15.12.2 Description

The **driver** element shall contain one of three different types of drivers (**defaultValue**, **clockDriver**, or **singleShotDriver**) that can be applied to a wire port of a component or abstractor.

- range** (optional) specifies the bit range of the port to which the given **driver** element is to be applied. If not specified, the **driver** applies to the entire port. See [C.7.14](#).
- viewRef** references a view of this component to which this driver applies. See [C.29](#).
- A **driver** shall also contain one of the following:
 - defaultValue** (type: *qualifiedExpression*) specifies a static value for this port. The **defaultValue** shall be defined to have a bit width that is compatible with the range of the port being driven. See [C.31](#) to determine when this value is applied.
 - clockDriver** specifies a repeating waveform for the specified driver range of this port. See [6.15.13](#).
 - singleShotDriver** specifies a non-repeating waveform for this port. When this element is contained within a non-scalar wire port, the single-shot waveform shall apply to all bits of this port. The **singleShotDriver** element contains three elements that describe the properties of the waveform. These are also depicted in [Figure 12](#).

- i) **singleShotOffset** (mandatory; type: *realExpression*, see [C.7.3](#)) specifies the time delay from the start of the waveform to the transition.

This element also contains a **units** (optional) attribute for specifying the units of the time values: **ns** (the default) and **ps**, where **ns** stands for nanosecond and **ps** stands for picosecond.

- ii) **singleShotValue** (mandatory; type: *unsignedBitVectorExpression*, see [C.7.10](#)) specifies the logic value to which the port transitions. This value is also the opposite of the value from which the waveform starts. The value of this element shall be 0 or 1.

- iii) **singleShotDuration** (mandatory; type: *realExpression*, see [C.7.3](#)) specifies how long the waveform remains at the value specified by **singleShotValue**.

This element also contains a **units** (optional) attribute for specifying the units of the time values: **ns** (the default) and **ps**, where **ns** stands for nanosecond, and **ps** stands for picosecond.

See also [SCR 6.23](#), [SCR 6.32](#), and [SCR 12.9](#).

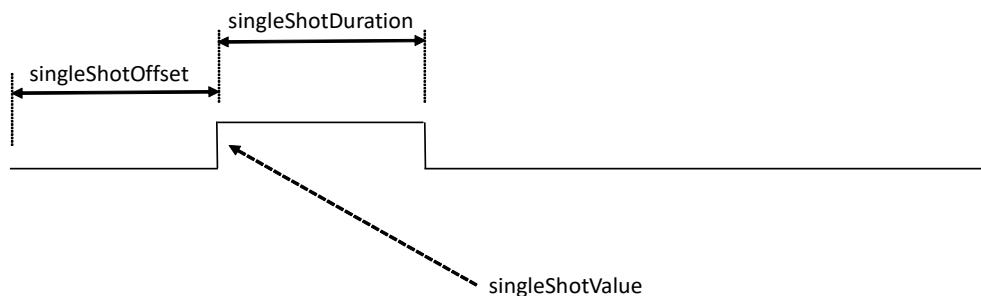
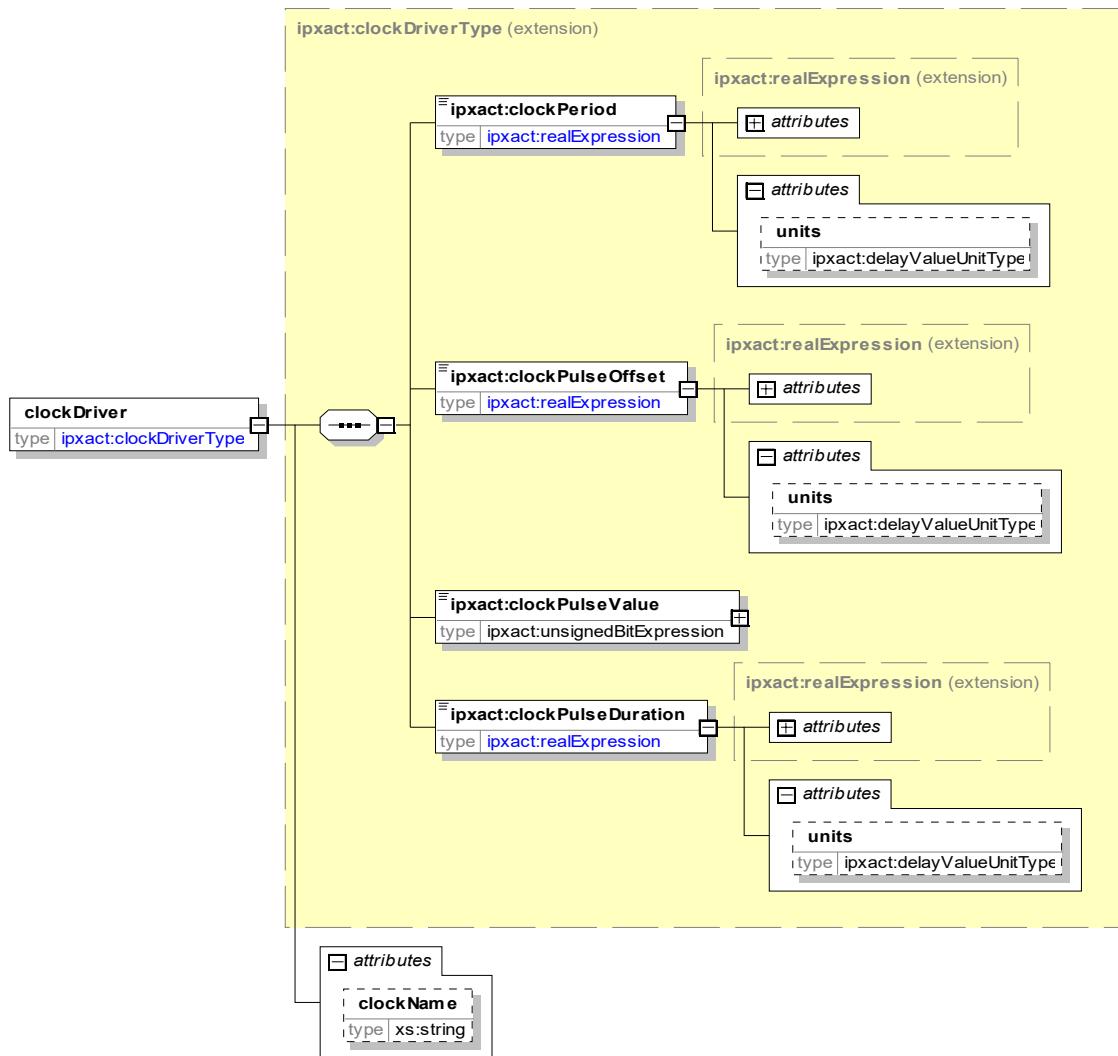


Figure 12—singleShotDriver elements

6.15.13 Component driver/clockDriver

6.15.13.1 Schema

The following schema details the information contained in the **clockDriver** element, which may appear as an element inside the top-level wire style **port/wire/driver** element. This element defines the properties of a clock waveform. When this element is contained within a non-scalar wire port, the clock waveform shall apply to all bits of this port.



6.15.13.2 Description

The `clockDriver` element contains four elements that describe the properties of a clock waveform (see also [Figure 13](#)). The optional `clockName` attribute (type: `string`) is used to specify a name for the clock being defined when it is different from the name of the enclosing port. A `clockDriver` contains all the following elements:

- clockPeriod** (mandatory; type: `realExpression`, see [C.7.3](#)) specifies the overall length (in time) of one cycle of the waveform. This element also contains a `units` (optional) attribute for specifying the units of the time values: `ns` (the default) and `ps`, where `ns` stands for nanosecond and `ps` stands for picosecond.
- clockPulseOffset** (mandatory; type: `realExpression`, see [C.7.3](#)) specifies the time delay from the start of the waveform to the first transition. This element also contains a `units` (optional) attribute for specifying the units of the time values: `ns` (the default) and `ps`, where `ns` stands for nanosecond and `ps` stands for picosecond.

- c) **clockPulseValue** (mandatory; type: *unsignedBitExpression*, see [C.7.9](#)) specifies the logic value to which the port transitions. This value is also the opposite of the value from which the waveform starts. The value of this element shall be 0 or 1.
- d) **clockPulseDuration** (mandatory; type: *realExpression*, see [C.7.3](#)) specifies how long the waveform remains at the value specified by **clockPulseValue**. This element also contains a **units** (optional) attribute for specifying the units of the time values: **ns** (the default) and **ps**, where **ns** stands for nanosecond and **ps** stands for picosecond.

See also [SCR 12.7](#) and [SCR 12.9](#).

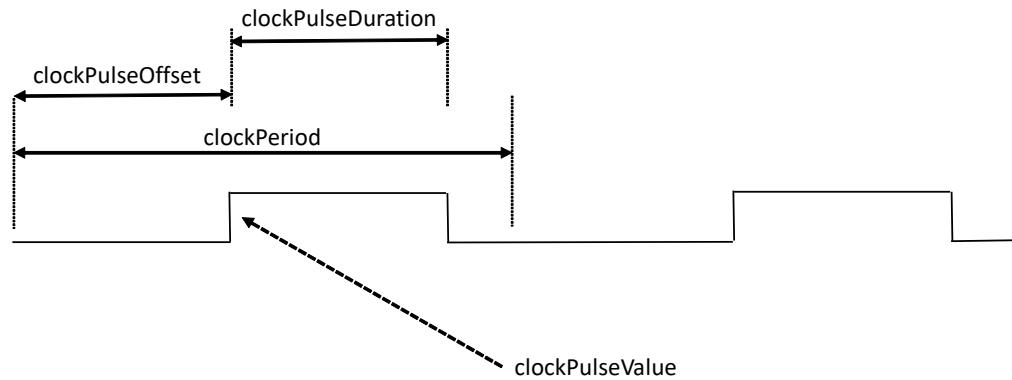


Figure 13—clockDriver elements

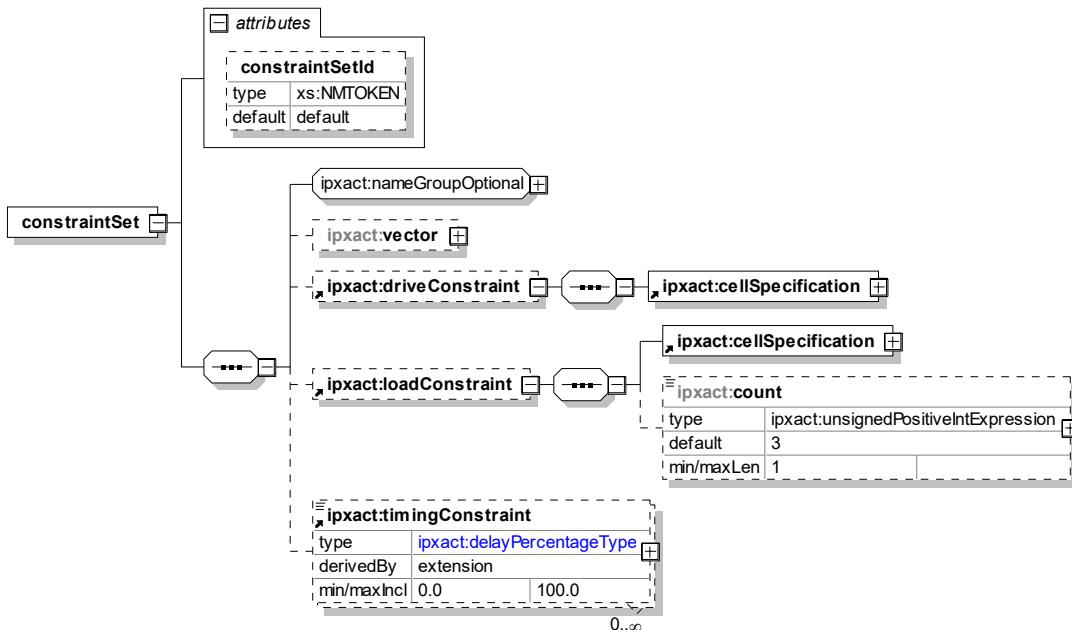
6.15.14 Implementation constraints

Implementation constraints can be defined to document requirements that need to be met by an implementation of the component. Constraints are defined in groups called *constraint sets* (in the IP-XACT element **port/wire/constraintSets/constraintSet**) so different constraints can be associated with different views of the component. A particular set of constraints is tied to a component view by the **constraintSetId** attribute in the constraint set and the matching **constraintSetRef** element in the view.

6.15.15 Component wire port constraints

6.15.15.1 Schema

The following schema defines the information contained in the **constraintSets** element, which may appear within a **wire** element within a component **port** element (**component/model/ports/port/wire**).



6.15.15.2 Description

The **constraintSets** element is used to define technology-independent implementation constraints associated with the containing wire port of the component. The **constraintSets** element contains one or more **constraintSet** elements that define a set of constraints for the port. If more than one **constraintSet** element is present, each shall have a unique value for the **constraintSetId** (mandatory; type: *NMTOKEN*; default: **default**) attribute so each **constraintSet** can be uniquely referenced from a **view**. **constraintSet** contains the following elements:

- nameGroupOptional** is defined in [C.19.3](#).
- vector** (optional) determines to which bits of a vectored port the constraint applies. The **left** and **right** vector bounds elements inside the **vector** element specify the bounds of the vector. The **left** and **right** elements are of type *nonNegativeInteger*.
- driveConstraint** (optional) defines a drive constraint for this port. The **driveConstraint** element may appear within a **modeConstraints** or **mirroredModeConstraints** element within a wire type port in an abstraction definition or within a **constraintSet** element within a wire type port in a component.

The **driveConstraint** element defines a technology-independent drive constraint associated with the containing wire port of a component or the component port associated with the logical port within an abstraction definition if the **driveConstraint** element is contained within an abstraction definition. The actual constraint consists of a technology-independent specification of a library cell presumed to drive the input port. The **cellSpecification** element defines the cell (see [6.15.17](#)).

The **driveConstraint** element is not valid on an output port.

- d) **loadConstraint** (optional) defines a load constraint for this port. The **loadConstraint** element may appear within a **modeConstraints** or **mirroredModeConstraints** element within a wire type port in an abstraction definition or within a **constraintSet** element within a wire type port in a component.

The **loadConstraint** element defines a technology-independent load constraint associated with the containing wire port of a component or the component port associated with the logical port within an abstraction definition if the **loadConstraint** element is contained within an abstraction definition. The actual constraint consists of two parts: the technology-independent specification of a library cell and a count. **loadConstraint** also contains the following elements:

- 1) **cellSpecification** (mandatory) defines the library cell (see [6.15.17](#)).
- 2) **count** (optional; type: *unsignedPositiveIntExpression* (see [C.7.13](#)); default: 3) indicates how many loads of the indicated type are modeled as if attached to the output port.

The **loadConstraint** element is not valid on input ports.

- e) **timingConstraint** (optional) defines a timing constraint relative to a clock for this port. See [6.15.16](#).

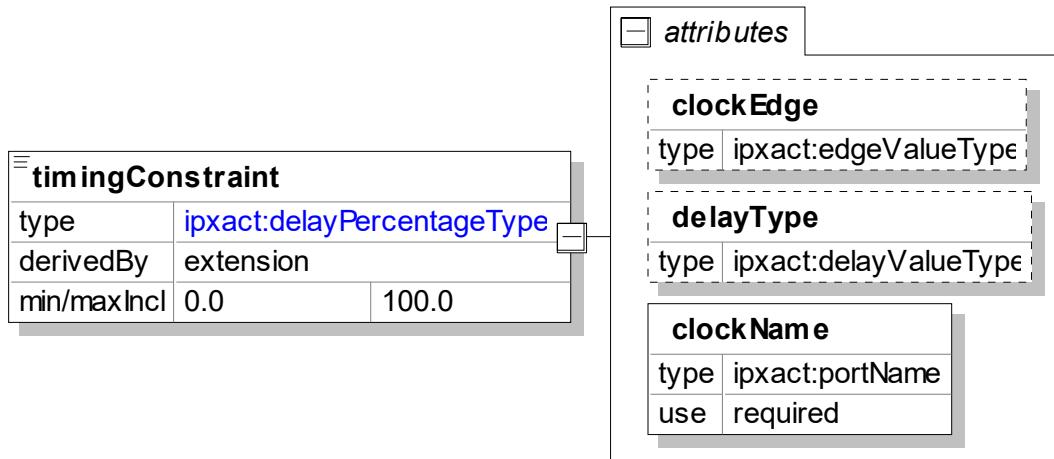
See also [SCR 12.1](#), [SCR 12.2](#), [SCR 12.3](#), [SCR 12.4](#), [SCR 12.5](#), and [SCR 12.6](#).

NOTE—To specify technology-dependent constraints (which are not represented in the schema), use an SDC file and reference the file via **fileSet**.

6.15.16 Port timing constraints

6.15.16.1 Schema

The following schema defines the information contained in the **timingConstraint** element, which may appear within a **modeConstraints** or **mirroredModeConstraints** element within a wire type port in an abstraction definition or within a **constraintSet** element within a wire type port in a component.



6.15.16.2 Description

The **timingConstraint** element defines a technology-independent timing constraint associated with the containing wire port of a component or abstraction definition. It is of type *delayPercentageType*; the value is a floating point number between 0 and 100, which represents the percentage of the cycle time to be allocated to the timing constraint on the port. If the component port is an input (or the port in an abstraction definition ends up mapping to a physical port with direction **in**), the timing constraint represents an input

delay constraint; otherwise, it represents an output delay constraint. **timingConstraint** also contains the following attributes:

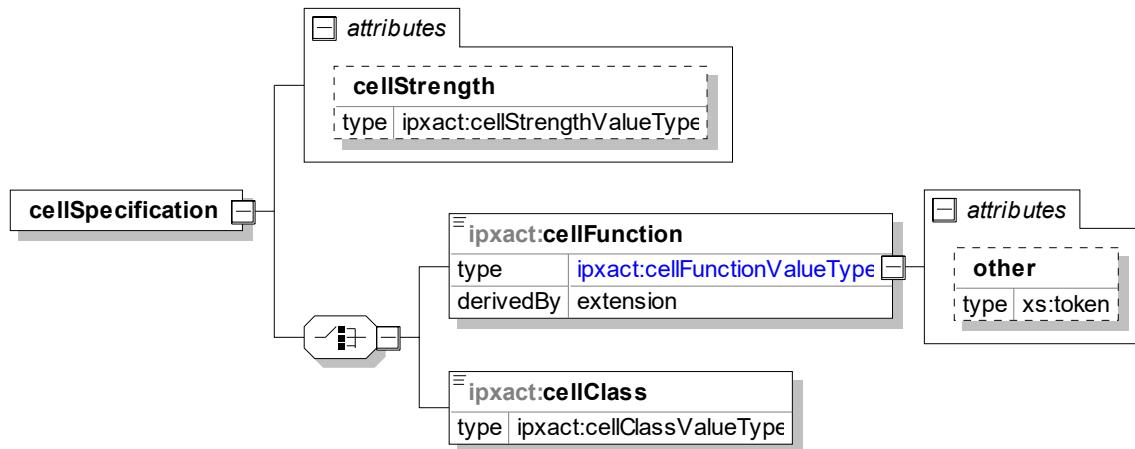
- clockEdge** (optional; default: **rise**) specifies to which edge of the clock the constraint is relative. The default behavior is that the constraint is relative to the rising edge of the clock. The **clockEdge** attribute may have two values **rise** or **fall**.
- delayType** (optional) restricts the constraint to applying to only best-case (minimum) or worst-case (maximum) timing analysis. By default, the constraint is applied to both. The **delayType** attribute may have two values **min** or **max**.
- clockName** (mandatory; type: *portName*) specifies the delay constraint relative to the clock. **clockName** shall be a valid port name or another clock name in the containing description. The cycle time of the referenced clock is what actually determines the actual magnitude of the delay constraint ($<\text{clock cycle time}> \times 100 / <\text{timing constraint element value}>$).

See also [SCR 12.7](#) and [SCR 12.8](#).

6.15.17 Load and drive constraint cell specification

6.15.17.1 Schema

The following schema defines the information contained in the **cellSpecification** element, which may appear within a **loadConstraint** or **driveConstraint** element indicating the type of cell to use in the constraint.



6.15.17.2 Description

The **cellSpecification** element defines a cell in a technology-independent fashion so that drive and load constraints can be defined without referencing a specific technology library. The cell is defined so a DE can map it to an appropriate cell in a specific library when the actual constraint is generated.

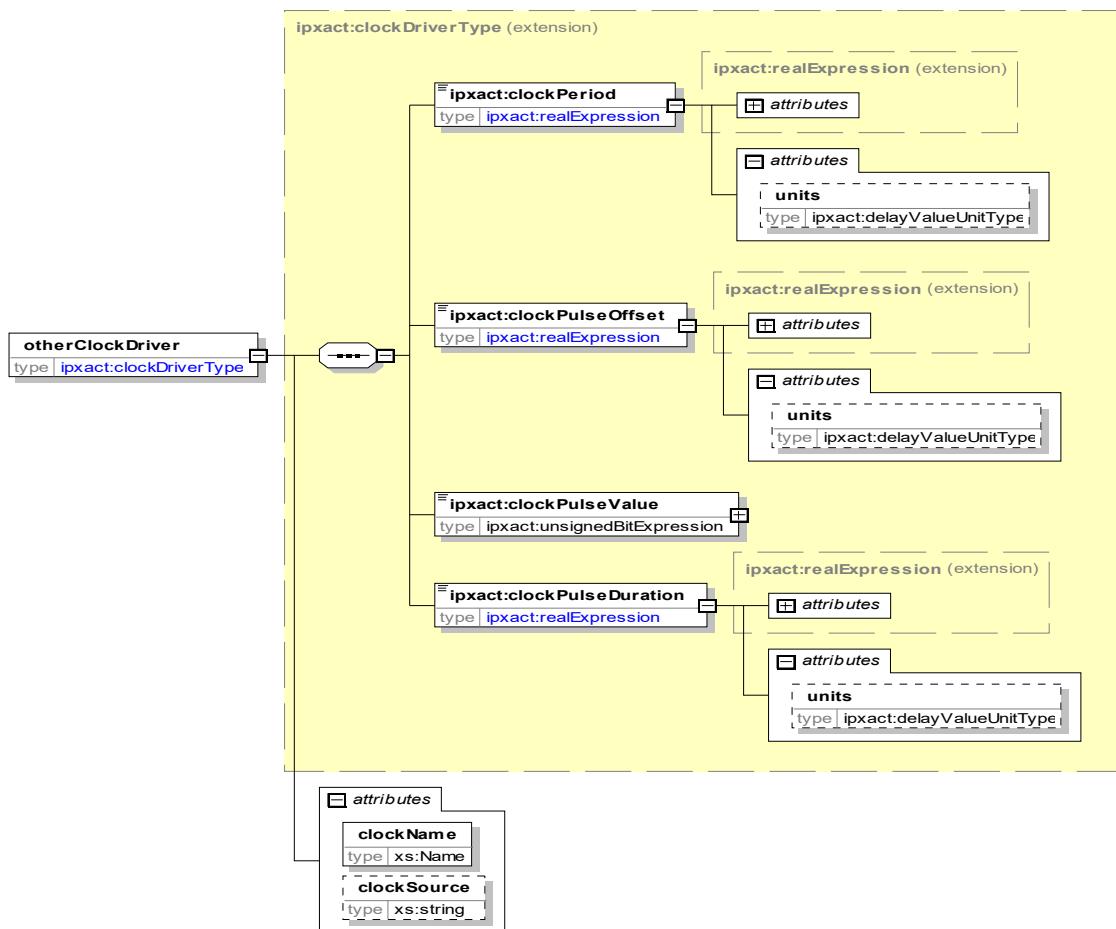
- cellStrength** (optional) specifies the desired strength for this cell.
- The **cellSpecification** element shall contain one of the following elements:
 - cellFunction** specifies a cell function from the user-defined library. The **cellFunction** element shall be one of the following values: **nd2**, **buf**, **inv**, **mux21**, **dff**, **latch**, **xor2**, or **other**.
 - The **cellFunction** element contains a **cellStrength** (optional; default: **median**) attribute that provides the cell strength specification. The value shall be one of **low**, **median**, or

- high.** **median** implies the middle cell of all the cells that match the desired function, sorted by drive or load strength (as appropriate for the given constraint), is used.
- ii) When the value **other** is used, the **other** attribute (type: **token**) can be set to a user-defined cell function. This attribute should be used only for documentation purposes as user-defined cell values are not guaranteed to be interpreted by DEs.
 - 3) **cellClass** specifies a cell class from the user-defined library. The **cellClass** element shall be one of the following values: **combinational** or **sequential**. The **cellClass** element contains a **cellStrength** (optional; default: **median**) attribute that provides the cell strength specification. The value shall be one of **low**, **median**, or **high**. **median** implies the middle cell of all the cells that match the desired class, sorted by drive or load strength (as appropriate for the given constraint), is used.

6.15.18 Other clock drivers

6.15.18.1 Schema

The following schema defines the information contained in the **otherClockDrivers** element, which may appear within a **component** element.



6.15.18.2 Description

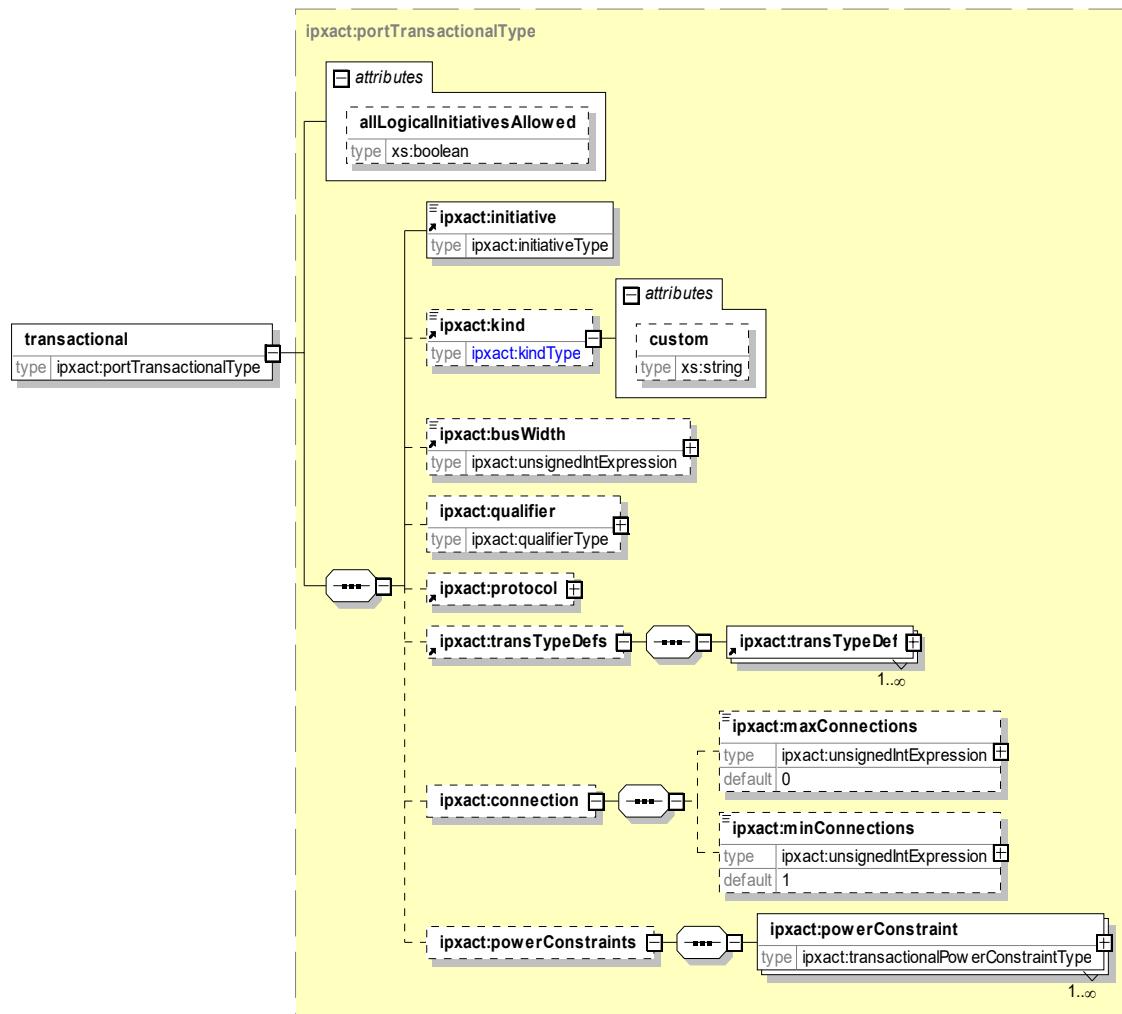
The **otherClockDrivers** element defines clocks within a component that are not directly associated with a top-level port, e.g., virtual clocks and generated clocks. The **otherClockDrivers** element contains one or more **otherClockDriver** elements, each of which represents a single clock. The **otherClockDriver** element consists of a number of subelements that define the format of the clock waveform.

- a) **clockPeriod**, **clockPulseOffset**, **clockPulseValue**, and **clockPulseDuration** (all required) are all detailed in the description of the element **clockDriver**. See [6.15.13](#).
- b) **clockName** (mandatory; type: *Name*) attribute indicating the name of the clock for reference by a constraint.
- c) **clockSource** (optional; type: *string*) attribute defines the physical path and name of the clock generation cell.

6.15.19 Component transactional port type

6.15.19.1 Schema

The following schema defines the information contained in the **transactional** element (in a **component/model/ports/port** element).



6.15.19.2 Description

A **transactional** element in a component model port defines a physical transactional port of the component, which implements or uses a service. A service can be implemented with functions or methods. It contains the following elements:

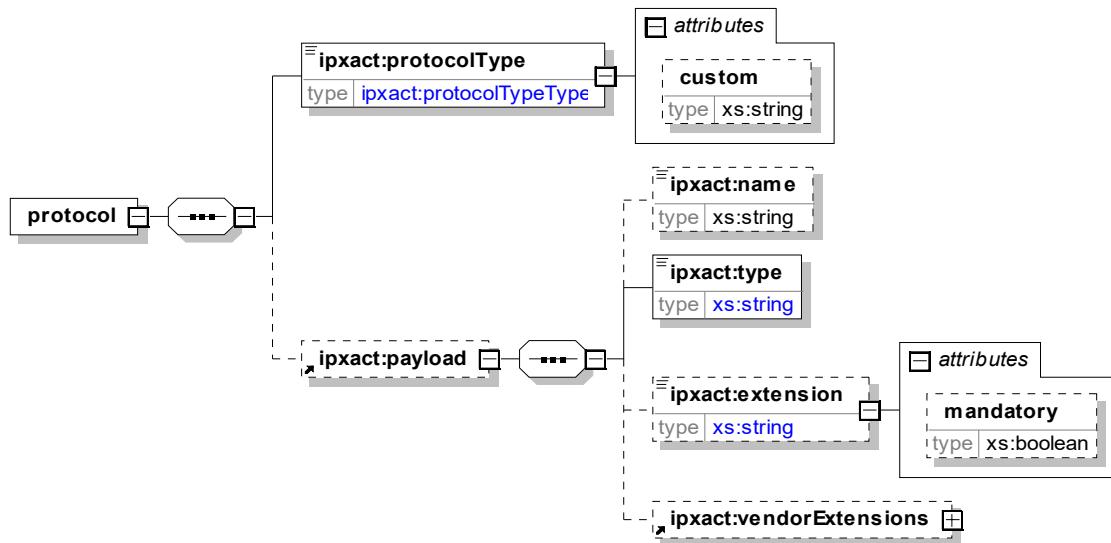
- a) **allLogicalInitiativesAllowed** (optional; type: *boolean*; default: **false**) attribute defines whether the port may be mapped to a port in an **abstractionDefinition** with a different initiative. See [5.3](#).
- b) **initiative** (mandatory) defines the type of access: **requires**, **provides**, **both**, or **phantom**.
 - 1) For example, a SystemC **sc_port** should be defined with the **requires** initiative, since it requires a SystemC interface. A SystemC **sc_export** should be defined with the **provides** initiative, since it provides a SystemC interface.
 - 2) **both** indicates the type of access is both **requires** and **provides**.
 - 3) **phantom** indicates a phantom port is being defined. See [6.15.21](#).
- c) **kind** (optional) specifies the transactional port's type. It can take one of the following enum values: **tlm_port**, **tlm_socket**, **simple_socket**, **multi_socket**, or **custom**. When **custom**, a *name* can be specified in the **custom** (type: *string*) attribute. Also, the **tlm_port** should be used only for TLM1 notation; the other enum values should be used for TLM2 sockets.
- d) **busWidth** (optional; type: *unsignedIntExpression*, see [C.7.11](#)) specifies the data width in bits, e.g., 32 or 64.
- e) **qualifier** (optional) indicates which type of information this wire port carries. See [5.6](#).
- f) **protocol** (optional) characterizes how the information is transported by the transactional port. See [6.15.20](#).
- g) **transTypeDefs** (optional) defines the port type expressed in the default language for this port. See [6.15.21](#).
- h) **connection** (optional) defines the number of legal connections for a port.
 - 1) **maxConnections** (optional; type: *unsignedIntExpression*, see [C.7.11](#); default: **0**) indicating the maximum number of connections that this port supports. 0 indicates an unbounded number of legal connections.
 - 2) **minConnections** (optional; type: *unsignedIntExpression*, see [C.7.11](#); default: **1**) indicating the minimum number of connections that this port supports.
- i) **powerConstraints** (optional) describes the port's power domain. See [C.24.1](#).

See also [SCR 6.2](#), [SCR 6.3](#), [SCR 6.4](#), [SCR 6.13](#), [SCR 6.21](#), and [SCR 6.22](#).

6.15.20 Component transactional protocol/payload definition

6.15.20.1 Schema

The following schema defines the information contained in the **protocol/payload** element (in a **component/model/ports/port/transactional** element).



6.15.20.2 Description

A **protocol** element characterizes the communication protocol. It contains a **protocolType** and (optionally) a **payload**. The **protocolType** can be **tlm** or **custom**.

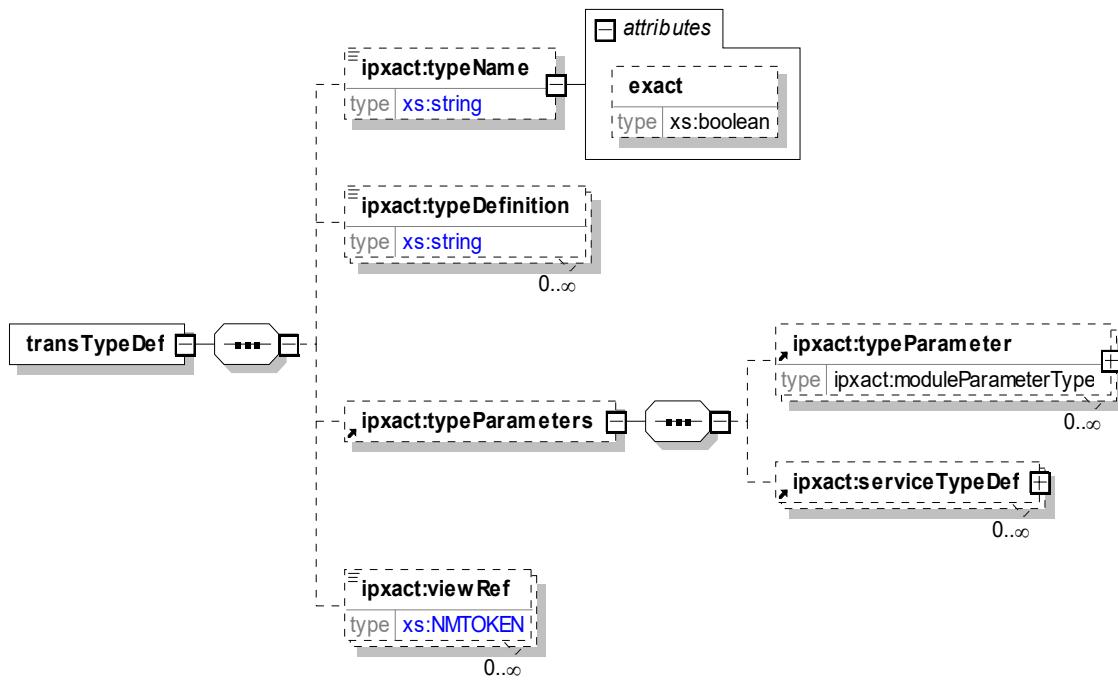
A **payload** element specifies how the information is transported. It contains the following subelements:

- name** (optional; type: *string*) specifies the payload's name.
- type** (mandatory; type: *string*) specifies if this is a **generic** or **specific** typing.
- extension** (optional; type: *string*; default: **optional**) determines if this payload extension is **mandatory** or **optional**.
- vendorExtensions** (optional) adds any extra vendor-specific data related to the service. See [C.27](#).

6.15.21 Component transactional port type definition

6.15.21.1 Schema

The following schema defines the information contained in the **transTypeDef** element (in a **component/model/ports/port/transactional** element).



6.15.21.2 Description

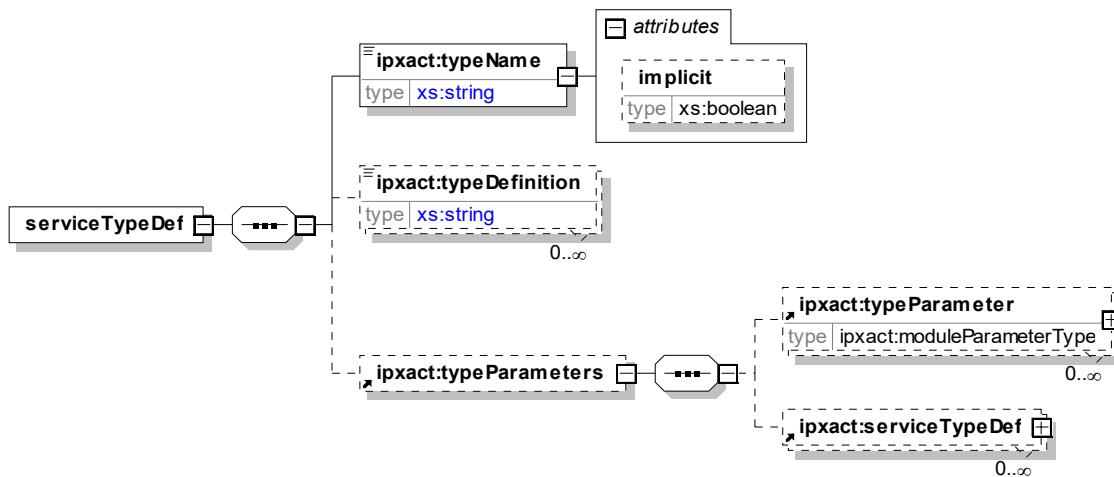
A **transTypeDef** element defines the port type expressed in the default language for this port (e.g., SystemC or SystemVerilog). It contains the following elements:

- typeName** (mandatory; type: *string*) defines the port type (such as `sc_port`/`sc_export` in SystemC or any user-defined type, such as `tlm_port`). The **typeName** element may be associated with an optional **boolean exact** attribute (default: `true`). If `false`, this indicates the port type is an abstract type that may not be related to an existing type in the language of the referenced view.
- typeDefinition** (optional; type: *string*) contains a language-specific reference to where the given type is actually defined. [Table 6](#) shows some examples. There can be multiple **typeDefinitions** for each port.
- typeParameters** (optional) specifies a list of port type parameters. See [6.15.22](#).
- viewRef** (optional; type: *NMTOKEN*) specifies the views to which the **transTypeDef** applies. See [C.29](#).

6.15.22 Component transactional port service

6.15.22.1 Schema

The following schema defines the information contained in the **serviceTypeDef** element (in a **component/model/ports/port/transactional/transTypeDefs/transTypeDef/typeParameters** element).



6.15.22.2 Description

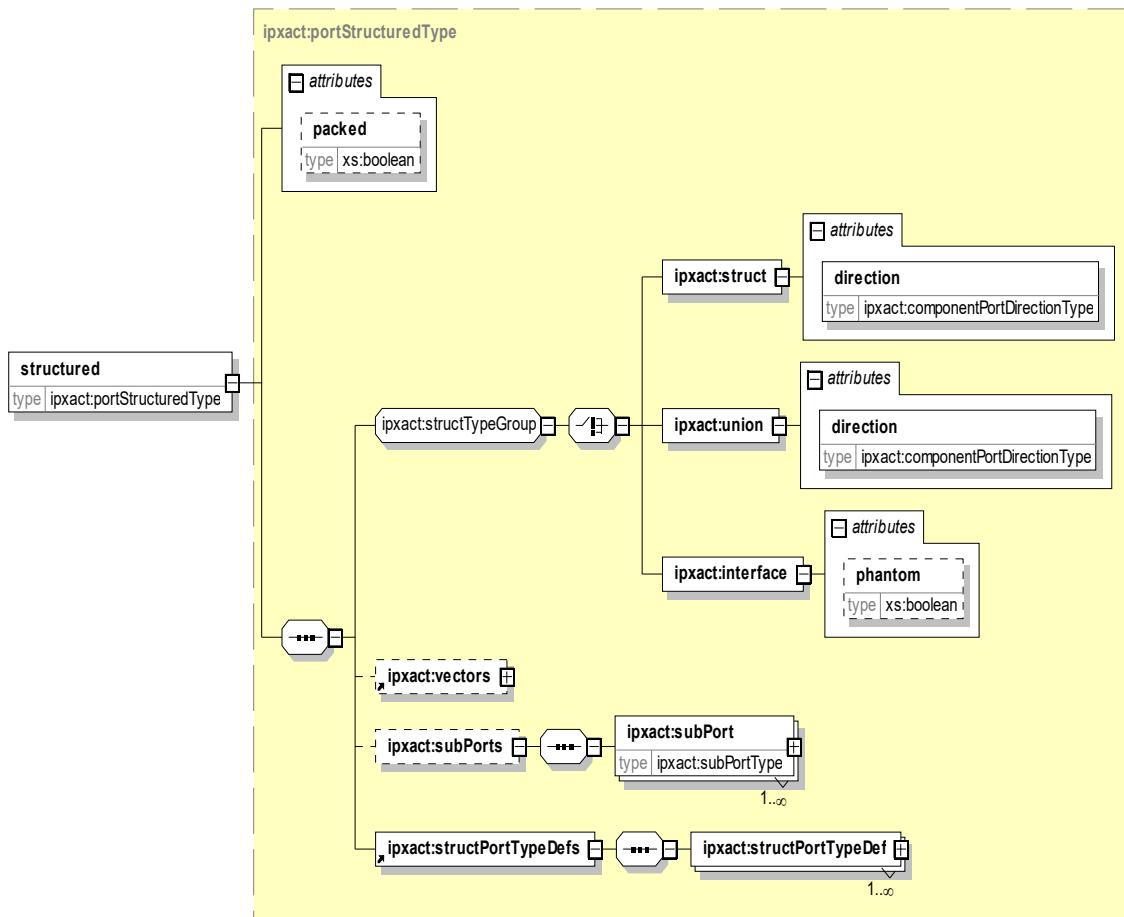
A **serviceTypeDef** element describes the interface protocol associated with the transactional port. It contains the following elements and attributes:

- typeName** (mandatory; type: *string*) defines the name of the service type (can be any predefined type, such as *boolean* or any user-defined type, such as *addr_type*). The **typeName** element may be associated with a *boolean implicit* attribute (default: *false*). If *true*, this indicates the service type should not be checked by SCRs.
- typeDefinition** (optional; type: *string*) indicates a location where the type is defined, e.g., in SystemC and SystemVerilog; this is the include file containing the type definition.
- typeParameters** (optional) specifies any service type parameters. **typeParameter** is identical to **moduleParameter** (see [6.15.6](#)). If a **typeParameter** describes the width of the transactional bus, the **typeParameter** element value shall match the **busWidth** element value in the transactional element (see [6.15.19.2](#)).

6.15.23 Component structured port type

6.15.23.1 Schema

The following schema defines the information contained in the element (in a **component/model/ports/port** element).



6.15.23.2 Description

A **structured** element in a component model port defines a physical port that represents a structure, union, or SystemVerilog interface. SystemVerilog interfaces are supported if they represent only structural connectivity. SystemVerilog interfaces that contain behavior are not supported:

- packed** (optional; type: `boolean`; default: `true`) attribute is also provided to identify if this structured port can be mapped to a packed bit sequence. Packed structured ports can be connected to other packed structured ports with the same or different typeName in `structPortTypeDefs` or to vectored wire ports of the same bit width. Unpacked structure ports can only be connected to other structured ports with the same typeName.
- structTypeGroup** (mandatory) is choice between elements `struct`, `union`, and `interface`. `Struct` indicates that the structured port is a structure of sub-ports. `Union` indicates that the structured port is union of subports. `Interface` indicates that structured port is an (SystemVerilog) interface. For packed structs, unions, and interfaces, the bits are organized as defined in SystemVerilog, IEEE Std 1800. Elements `struct` and `union` have an attribute `direction`. The value of `direction` shall match with

the direction of the wire sub-ports of the structured port. Element **interface** has an attribute **phantom** (optional; type **boolean**: default **false**). The value of phantom indicates if the interface is a phantom port. See [6.15.26](#).

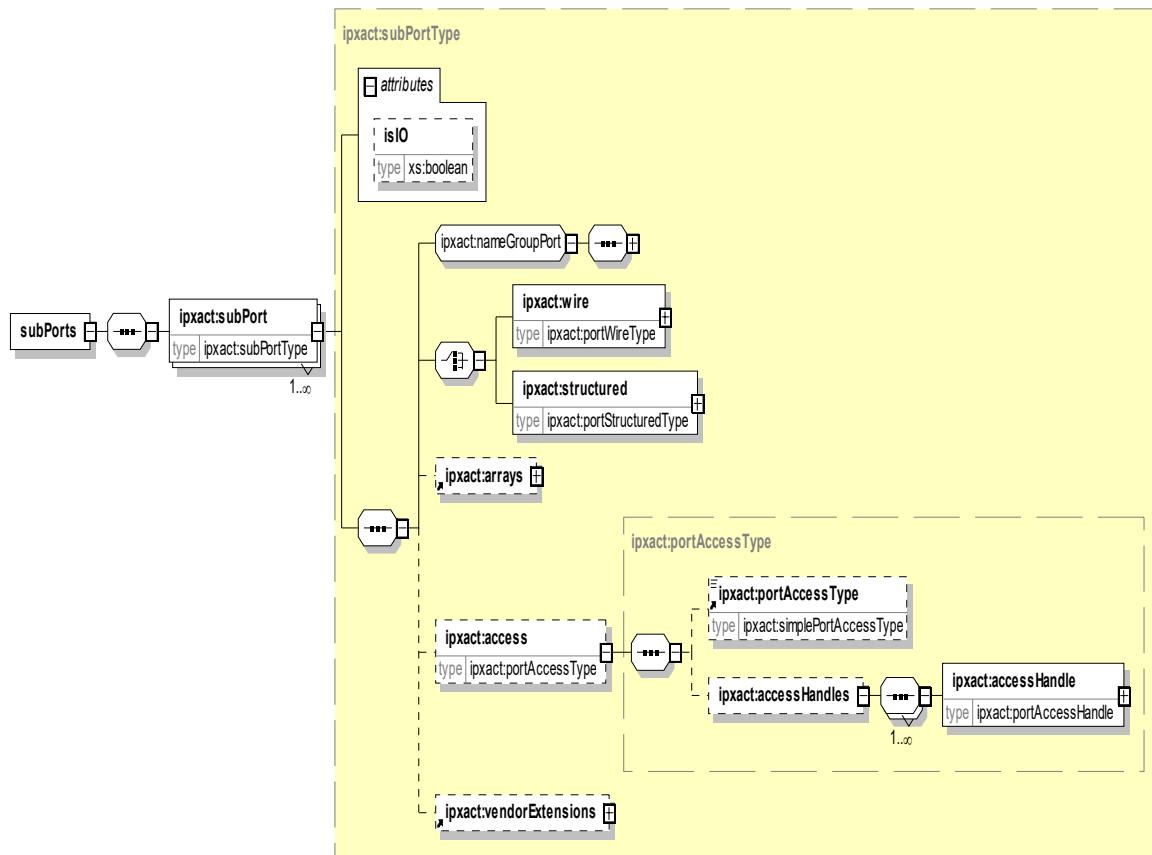
- c) **vectors** (optional) define the dimensions of a **vector** when the port represents a packed array or bit-vector. The **vector** element contains left and right elements. Vectors are also in component wire ports and abstractor ports. See [C.26.2](#).
- d) **subPorts** (optional) contains a **nameGroup**, a choice between **wire** and **structured**, **arrays**, **access**, and **vendorExtensions**. See [6.15.24](#).
- e) **structPortTypeDefs** (mandatory) element contains one or more **structPortTypeDef** elements describing details of this structured port. See [6.15.25](#).

See also [SCR 6.28](#), [SCR 6.57](#).

6.15.24 Subports

6.15.24.1 Schema

The following schema defines the information contained in the element (in a **component/model/ports/port** element).



6.15.24.2 Description

A **subport** describes a sub-element of a structured port. If the encapsulating structured port is a structure or a union, the subport describes a field of that struct or union. If the encapsulating structured port is an interface, then the subport describes a signal, a port, or a modport of that interface, depending on the role

element value. The **subport** element contains the following sub-elements and attributes:

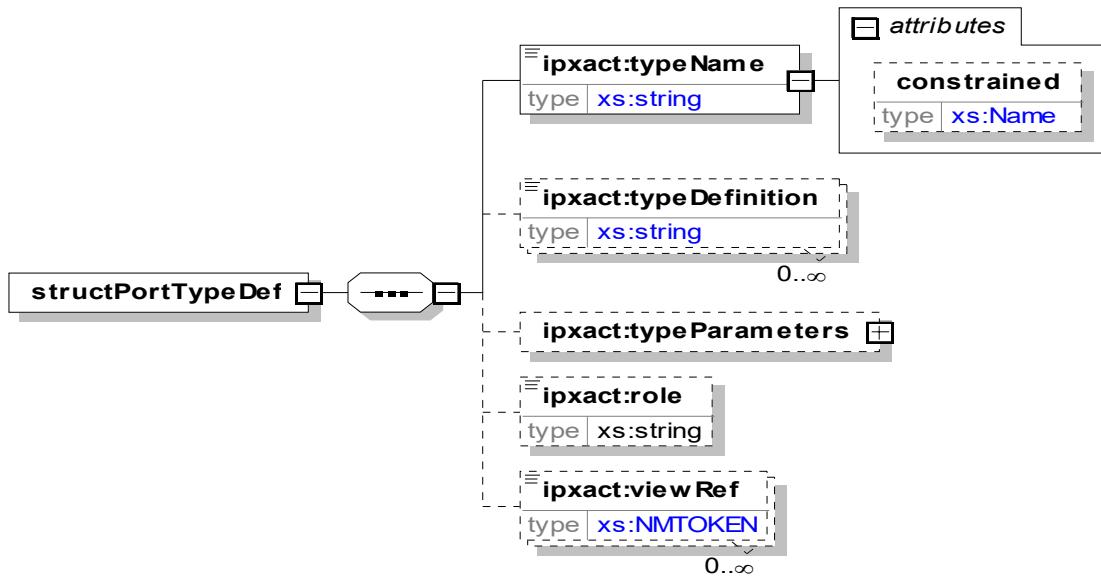
- a) **isIO** attribute indicates that a port is an input, output, or inout port on the SystemVerilog interface.
- b) **nameGroupPort** group elements shall be unique within the containing ports element. See [C.19.4](#).
- c) **wire** (mandatory) defines ports that transport purely binary values. Values may be aggregated using vectors and/or arrays. See [6.15.8](#).
- d) **structured** (mandatory) defines a port that is a structure, union, or SystemVerilog interface composed out of binary values. See [6.15.23](#).
- e) **arrays** (optional type: *configurableArrays*) specifies dimensions for a port defined as an array. See [C.4.2](#).
- f) **access** (optional) defines the access for a port.
 - 1) **portAccessType** (optional; default: **ref**) indicates to a netlister how to access the port. The **portAccessType** has one of two possible values **ref** or **ptr**. If **ref**, a netlister should access the port directly (i.e., by reference), and if **ptr**, it should access the port with a pointer.
 - 2) **accessHandles** (optional) indicates to a netlister the method to be used to access the object representing the port when references need to be done using something other than the simple port name. This is typically a function call. The **accessHandles** element contains a list of **accessHandle** elements of type **portAccessHandle**. See [C.1.3](#).
- g) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See [C.27](#).

See also [SCR 6.50](#) and [SCR 6.53](#).

6.15.25 Struct Port Type

6.15.25.1 Schema

The following schema defines the information contained in the element (in a **component/model/ports/port** element).



6.15.25.2 Description

A structPortTypeDefs element contains one or more structs:

- a) **typeName** (optional; type: *string*) defines the name of the type for the port. For VHDL, some typical values would be std_logic and std_ulogic. See [6.15.21](#).
- 1) **constrained** (optional; type: *list of strings*) attribute indicates for which vectors and arrays the number of bits in the type declaration is fixed by referencing the **vectorId** and **arrayId** values. The **vectorId** and **arrayId** values are separated by white space. If the number of bits is fixed (i.e., **vectorId** or **arrayId** appear in the constrained list), the bit indices are not required when referencing the type. When **constrained** is not present, bit indices are required on all references to the module parameter. See [6.15.11.2.1](#).
- b) **typeDefinition** (optional) documents the existence of **typeDefinitions** in a component.
- c) **typeParameters** (optional) specifies a list of port type parameters. See [6.15.22](#).
- d) **role** (optional) contains indicates the role that this structured port plays in a connection. Only valid for structured ports with sub-element interface. In SystemVerilog, the role indicates the modport for the SystemVerilog interface corresponding to this **structuredPortTypeDef**.
- e) **viewRef** (optional; type: **NMOKEN**) specifies the views to which the **structPortTypeDef** applies. See [C.29](#).

See also [SCR 6.51](#) and [SCR 6.52](#).

6.15.26 Phantom ports

In some components, the RTL or TLM implementation of the component does not fully implement the functionality of the component described by IP-XACT. In RTL components, this is typically because the component has to work in design flows that allow a signal to be routed through an RTL component only if there is some logic within the RTL component associated with that signal. This is particularly a problem for components containing channels or bridges.

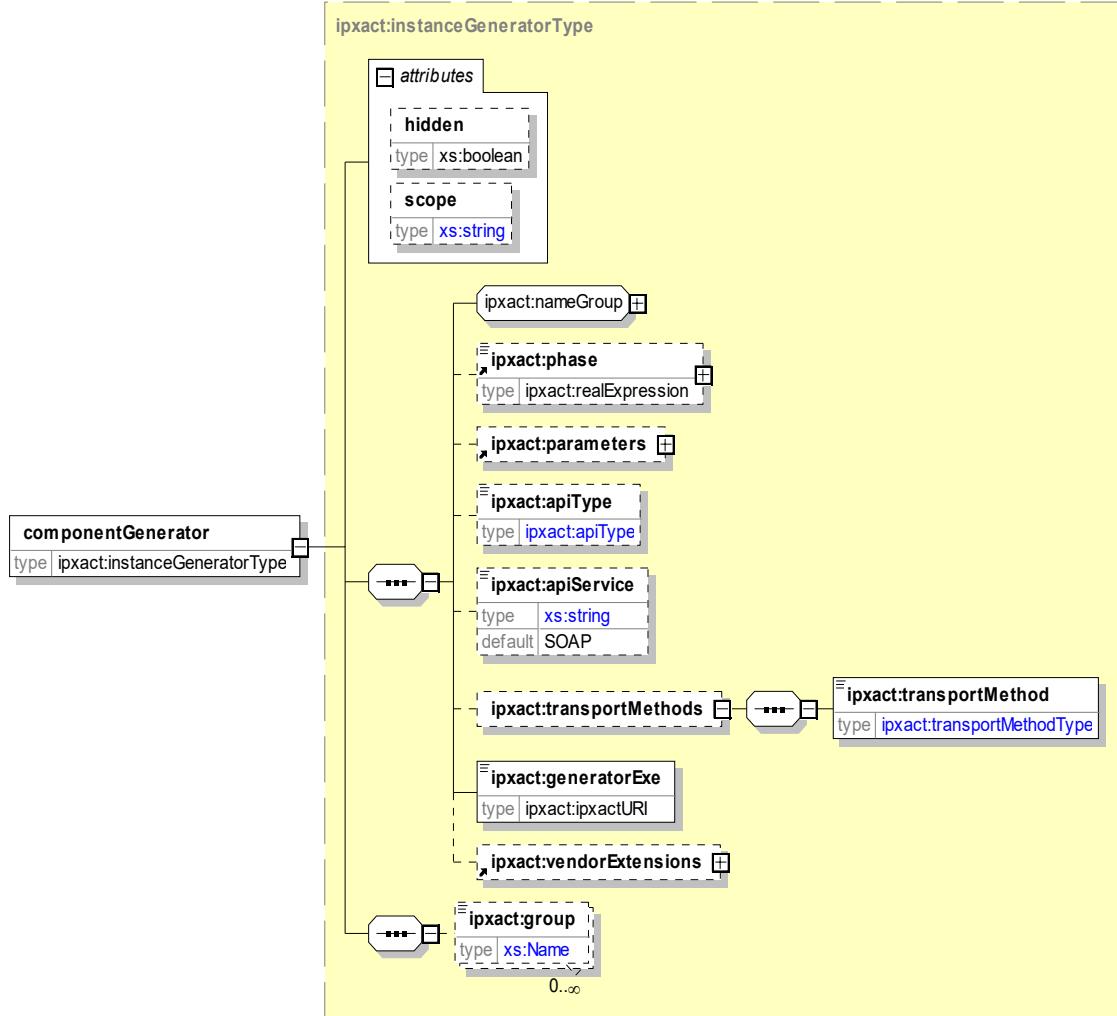
An IP-XACT channel or bridge can represent the complete bus infrastructure between the initiator, target, and system bus interfaces connected to the bus. As such, the component containing the channel should contain everything that is needed to create this infrastructure. In many buses, however, some signals are directly connected between the components attached to the bus, with no intervening logic. This is most often the case with clock and reset signals. If the component is to be usable in a wide range of design flows, these signals cannot be included in the RTL of the component.

To fully describe such a channel or bridge component and allow netlisters that have no special knowledge of that bus type to netlist designs containing it, IP-XACT describes these additional connections as phantom ports. *Phantom ports* are additional ports included in the component's port list, but marked as **phantom**. As with real component ports, the mapping of a set of logical bus ports to that phantom port implies any design using that component needs to connect those logical ports with no intervening logic. The difference is a real component port needs to have a corresponding port in any RTL, TLM, or hierarchical IP-XACT implementation of the component; whereas, for phantom ports there is no corresponding port in the implementation.

6.16 Component generators

6.16.1 Schema

The following schema details the information contained in the **componentGenerators** element, which may appear as an element inside the top-level **component** element.



6.16.2 Description

The **componentGenerators** element contains an unbounded list of **componentGenerator** elements. Each **componentGenerator** element defines a generator that is assigned and may be run on this component. **componentGenerator** contains the following attributes and elements:

- The **hidden** (optional; type; **boolean**; default: **false**) attribute specifies, when **true**, this generator shall not be run as a stand-alone generator and is required to be run as part of a chain. This generator should not be presented to the user for direct invocation. If **false**, this generator may be run as a stand-alone generator or in a generator chain.
- The **scope** (optional; type; **string**; default: **instance**) attribute is an enumerated list of **instance** and **entity**. **instance** indicates this generator shall be run once for all instances of this component. **entity** indicates this generator shall be run once for each instance of this component.

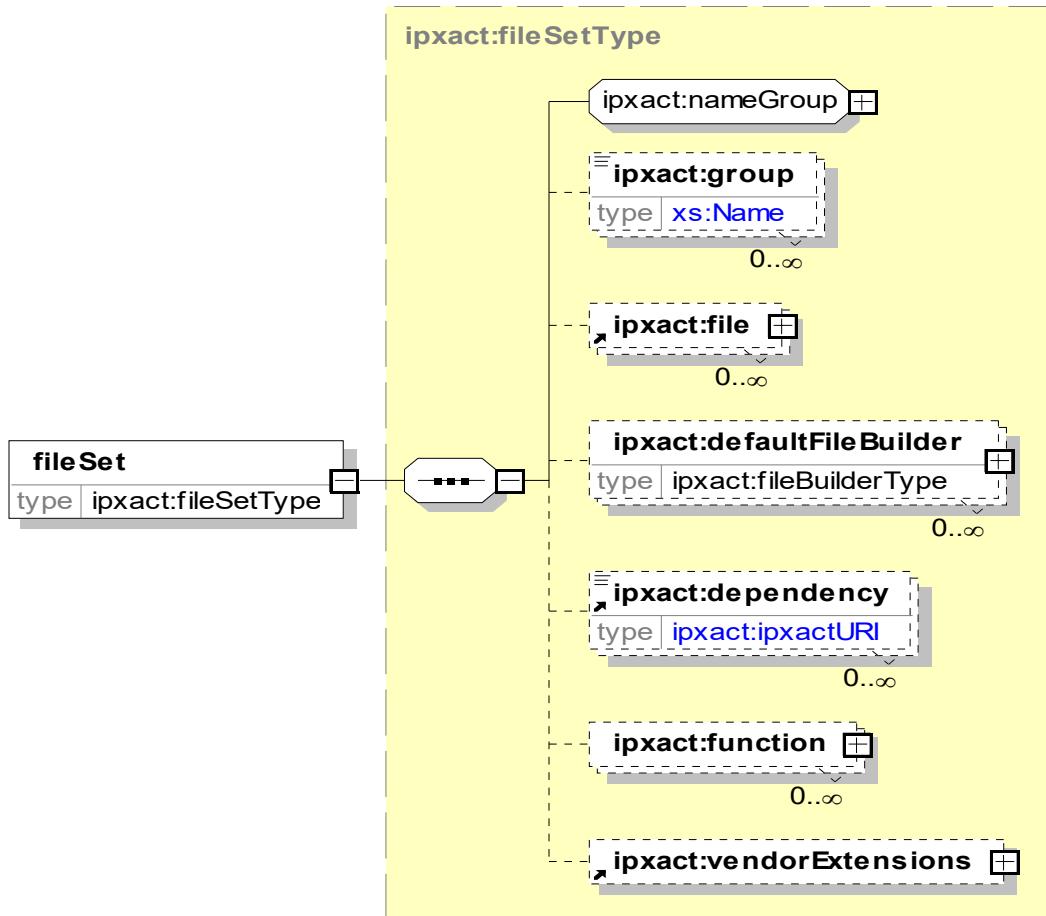
- c) **nameGroup** group is defined in [C.19](#). The **name** elements shall be unique within the containing **componentGenerators** element.
- d) **phase** (optional; type: *realExpression*, see [C.7.3](#)) determines the sequence in which generators are run. Generators are run in order starting with zero (0). If two generators have the same phase number, the order shall be interpreted as not important, and the generators can be run in any order. If no phase number is given, the generator is considered in the “last” phase, and these generators are run in any order after the last generator with a phase number. The **phase** element also shall be a positive number.
- e) **parameters** (optional) specifies any **componentGenerator** parameters. See [C.21](#).
- f) **apiType** (optional, type: **apiType**, default: **TGI_2022_BASE**) indicates the type of application programmers interface (API) used by the generator. Allowed values include the following:
 - 1) **TGI_2022_BASE** indicates a generator that communicates using the base TGI API defined for this standard.
 - 2) **TBGI_2022_EXTENDED** indicates a generator that communicates using the base and extended TGI APIs defined for this standard.
 - 3) **TGI_2014_BASE** indicates a generator that communicates using the base TGI API defined for IEEE Std 1685-2014. This API is not part of this standard and does not need to be supported by IP-XACT enabled DES.
 - 4) **TGI_2014_EXTENDED** indicates a generator that communicates using the base and extended TGI APIs defined for IEEE Std 1685-2014. These APIs are not part of this standard and do not need to be supported by IP-XACT enabled DES.
 - 5) **TGI_2009** indicates a generator that communicates using the TGI API defined for IEEE Std 1685-2009. This API is not part of this standard and does not need to be supported by IP-XACT-enabled DEs.
 - 6) **none** indicates the generator does not communicate with the DE using an API defined in this standard.
- g) **apiService** (optional) describes which transport service is used, i.e., SOAP or REST. Default is SOAP.
- h) **transportMethods** (optional) defines alternate transport protocols that this generator can support. The default transport protocol is HTTP if this element is not present.
transportMethod specifies an alternate transport protocol. This element is an enumerated list of only one element **file**.
 - i) **generatorExe** (mandatory; type: *ipxactURI*) contains an absolute or relative (to the location of the containing document) path to the generator executable. The path may also contain environment variables from the host system, which are used to abstract the location of the generator.
 - j) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **component-Generator**. See [C.27](#).
 - k) **group** (optional; type: *Name*) is an unbounded list of names used to assign this generator to a group with other generators. These **group** names are then referenced by a generator chain selector to forming a chain of generators. See [10.1](#).

6.17 File sets

6.17.1 fileSets

6.17.1.1 Schema

The following schema details the information contained in the **fileSets** element, which may appear in a component or an abstractor.



6.17.1.2 Description

The **fileSets** element contains one or more **fileSet** elements. A **fileSet** contains a list of files and directories associated with a component and/or instructions for further processing. If compilation order is important (e.g., for VHDL files), the files shall be listed in the order needed for compilation (the files to compile first are listed first). **fileSet** has the following mandatory and optional elements:

- nameGroup** group is defined in [C.19](#). The **name** elements shall be unique within the containing **fileSets** element.
- group** (optional; type: *Name*) describes the function or purpose of the file set with a single unbounded word group name (e.g., diagnostics, interrupt).
- file** (optional) references a single unbounded file or directory associated with the file set. If compilation order is important (e.g., for VHDL files), the files shall be listed in the order needed for compilation (see [6.17.2](#)).

- d) **defaultFileBuilder** (optional) specifies the unbounded default build commands for the files within this file set. See [6.17.4](#).
- e) **dependency** (optional; type: *ipxactURI*) is the path to a directory containing (include) files on which the file set depends.
- f) **function** (optional) specifies the unbounded information about a software function for a generator (see [6.17.5](#)).
- g) **vendorExtensions** (optional) provides a place for any vendor-specific extensions. See [C.27](#).

6.17.2 file

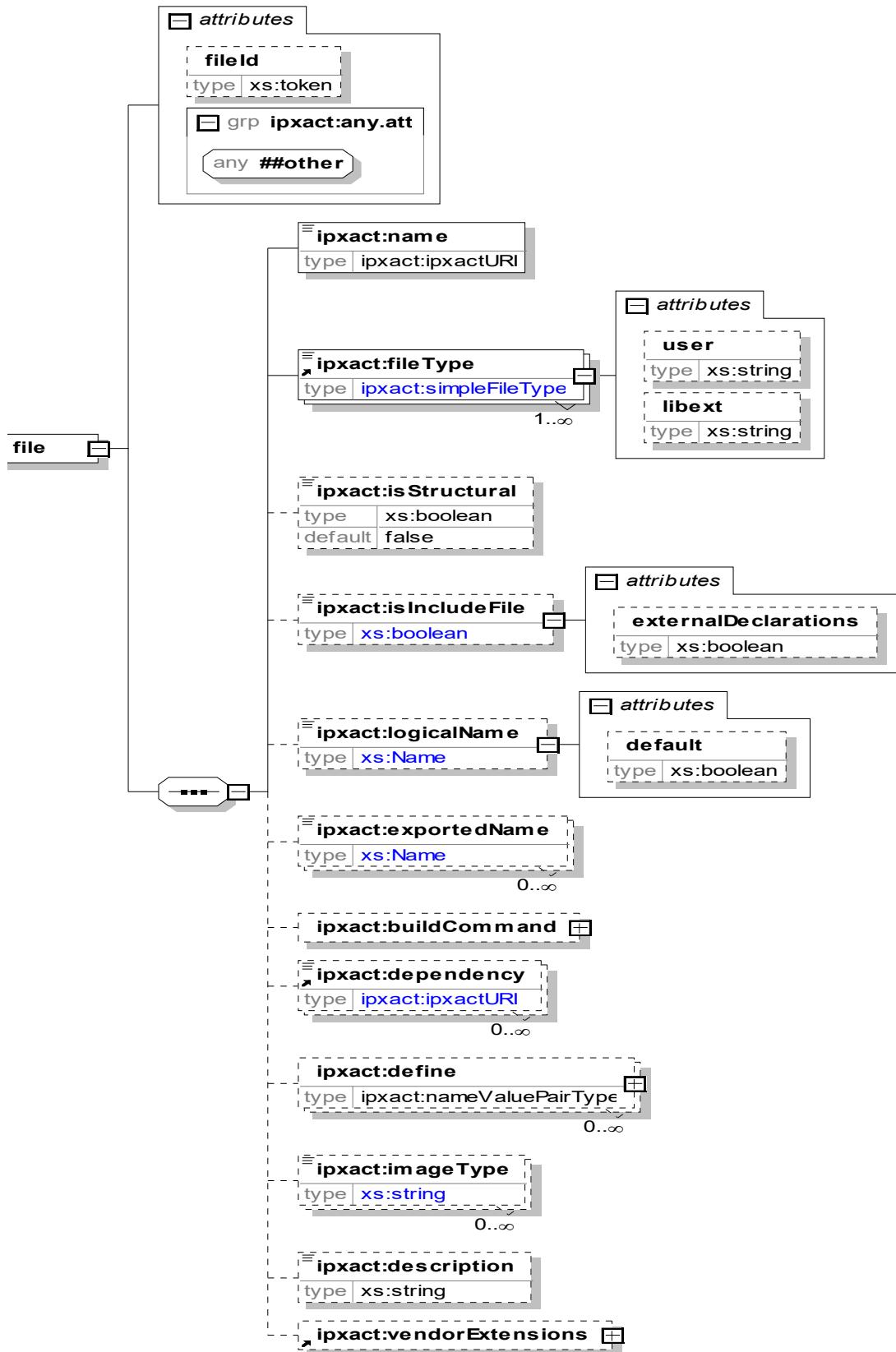
6.17.2.1 Schema

The following schema details the information contained in the **file** element, which may appear as an element inside the **fileSet** element.

6.17.2.2 Description

A **file** is a reference to a file or directory. It is an optional element of a **fileSet**. If compilation order is important (e.g., for VHDL files), the files shall be listed in the order needed for compilation (the files to compile first are listed first). The **file** element contains an attribute **fileId** (optional; type: *token*) that is used for references to this file from inside the **fileSet/function/fileRef** element. The **file** element also allows for vendor attributes to be applied. **file** contains the following elements:

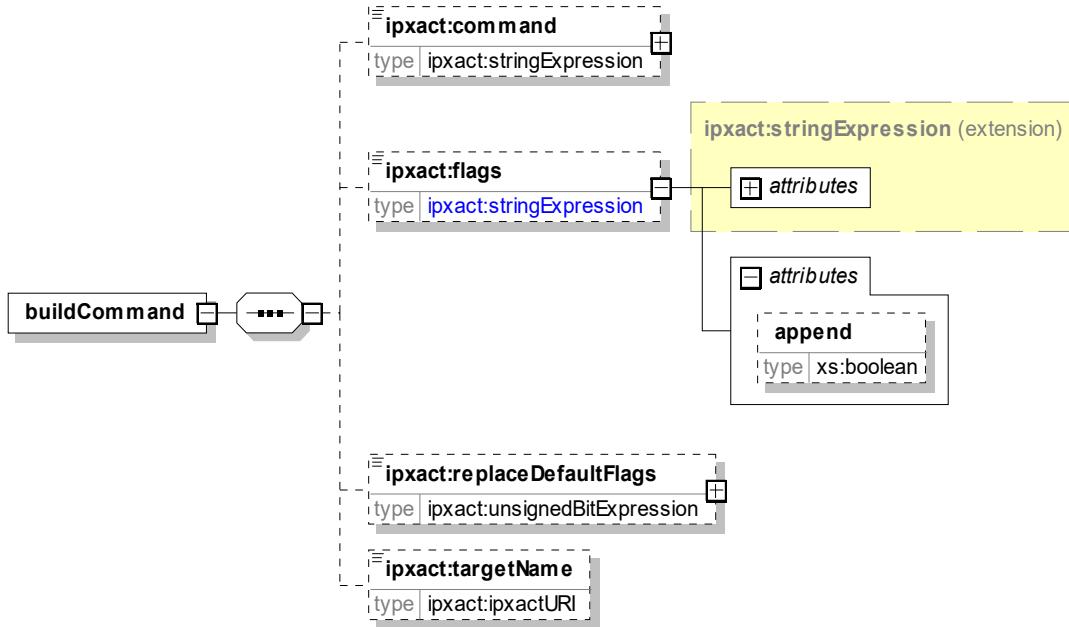
- a) **name** (mandatory; type: *ipxactURI*, see [D.11](#)) contains an absolute or relative (to the location of the containing document) path to a file name or directory. The path may also contain environment variables from the host system in the form of \${ENV_VAR}, used to abstract the location of files. See also [D.11](#).
- b) **fileType** (mandatory) group contains one or more of the elements defined in [C.16](#).
- c) **isStructural** (optional; type: *boolean*; default: **false**) indicates this file contains only structural RTL. This is a content hint for any tools processing this file.
- d) **includeFile** (optional; type: *boolean*), when **true**, declares the file as an include file. If this element is not present the default value is **false**. **includeFile** has an attribute **externalDeclarations** (optional; type: *boolean*; default: **false**); when **true**, this indicates the include file is needed by users of any files in this file set.
- e) **logicalName** (optional; type: *Name*) is the logical name for the file or directory, such as a VHDL library. **logicalName** includes an attribute **default** (optional; type: *boolean*; default: **false**) that means (when **true**) the logical name shall be used only as a default and another process may override this name. If **false**, this logical name shall always be used.
- f) **exportedName** (optional; type: *Name*) defines any names that can be referenced externally.
- g) **buildCommand** (optional) contains flags or commands for building the containing source file. These flags or commands override any flags or commands present in higher-level **defaultFileBuilder** elements. See [6.17.3](#).
- h) **dependency** (optional; type: *ipxactURI*) is the path to a directory containing (include) files on which the file depends.
- i) **define** (optional; type: *nameValuePairType*, see [C.20](#)) specifies the define symbols to use in the source file. This **define** element allows vendor attributes to be applied.
- j) **imageType** (optional; type: *string*) relates the current file to an executable image type in the design.
- k) **description** (optional; type: *string*) details the file for the user.
- l) **vendorExtensions** (optional) provides a place for any vendor-specific extensions. See [C.27](#).



6.17.3 buildCommand

6.17.3.1 Schema

The following schema details the information contained in the **buildCommand** element, which may appear as an element inside the **file** element.



6.17.3.2 Description

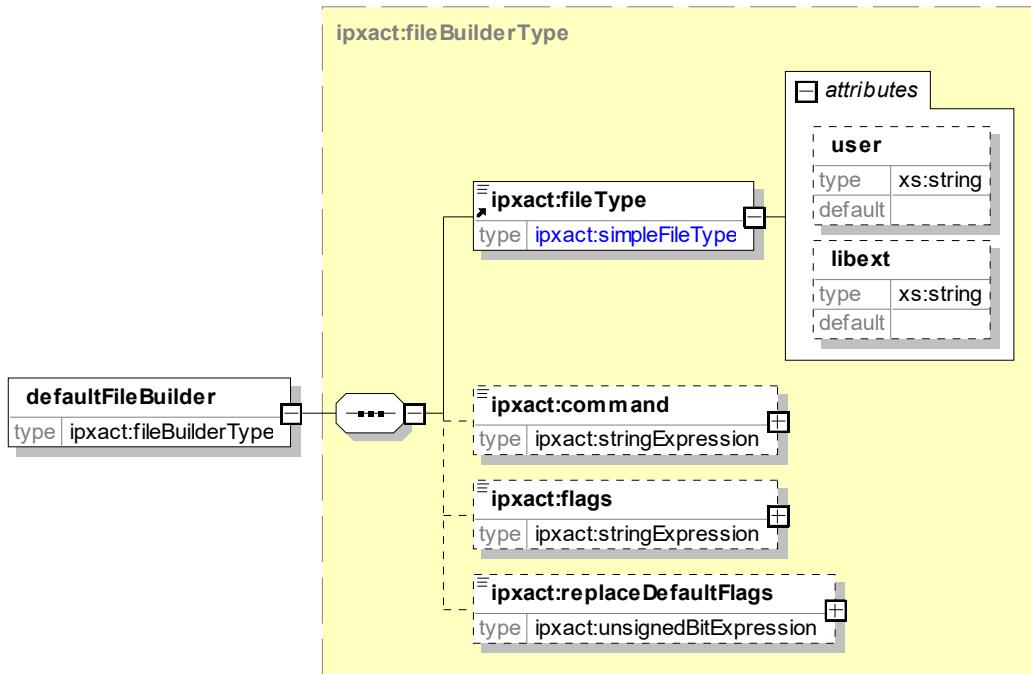
A **buildCommand** contains flags or commands for building the containing source file. These flags or commands override any flags or commands present in higher-level **defaultFileBuilder** elements.

- command** (optional; type: *stringExpression*, see [C.7.5](#)) element defines a compiler or assembler tool that processes files of this type.
- flags** (optional; type: *stringExpression*, see [C.7.5](#)) documents any flags to be passed along with the software tool command. The **flags** element contains an attribute **append** (optional; type: *boolean*), which when **true**, indicates the **flags** shall be appended to the current flags from the **defaultFileBuilder** (see [6.17.4](#)), **fileBuilder** (see [6.20.5](#)), or the build script generator. If **false**, the **flags** shall replace the existing flags.
- replaceDefaultFlags** (optional; type: *unsignedBitExpression*, see [C.7.9](#)), when **1**, documents flags that replace any of the default flags from the build script generator. If **0**, the flags are appended. If **1** and the **flags** element is empty or not present, this has the effect of clearing all the flags. If this element is not present, its effective value is **0**.
- targetName** (optional; type: *ipxactURI*, see [D.11](#)) defines the path to the file derived from the source file.

6.17.4 defaultFileBuilder

6.17.4.1 Schema

The following schema details the information contained in the **defaultFileBuilder** element, which may appear as an element inside the **fileSet** or **view** element.



6.17.4.2 Description

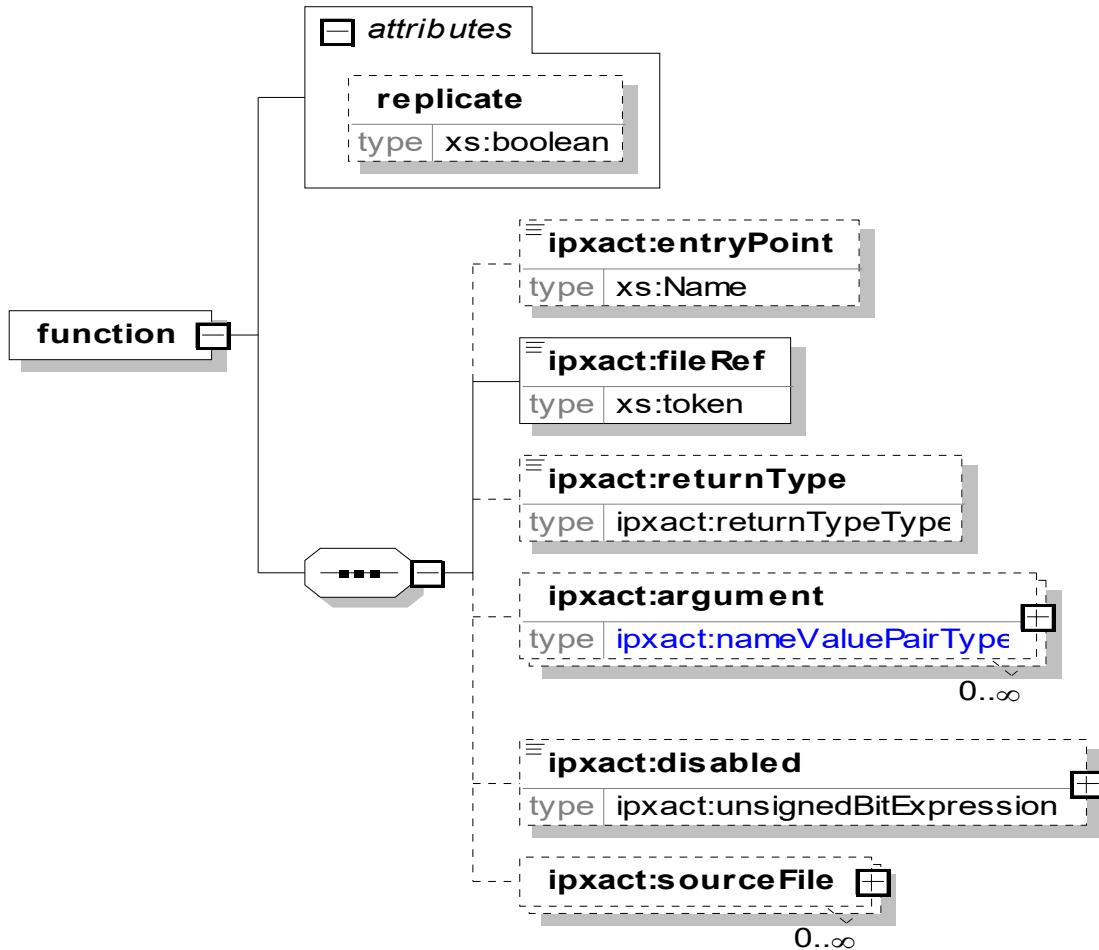
A **defaultFileBuilder** contains default flags or commands for building the containing source file types. These flags or commands may be overridden by flags or commands present in lower-level **defaultFileBuilder** or **buildCommand** elements.

- fileType** (mandatory) group contains one or more of the elements defined in [C.16](#). If the **fileType** is set to **other**, the optional attribute **other** (type: **string**) can be set to indicate a user-defined file type. No semantic meaning is inferred from the user-defined file type.
- command** (optional; type: **stringExpression**, see [C.7.5](#)) element defines a compiler or assembler tool that processes files of this type.
- flags** (optional; type: **stringExpression**, see [C.7.5](#)) documents any flags to be passed along with the software tool command.
- replaceDefaultFlags** (optional; type: **unsignedBitExpression**, see [C.7.9](#)) when **0** indicates the **flags** shall be appended to the current flags. If **1**, the **flags** shall replace the existing flags. If this element is not present, its effective value is **0**.

6.17.5 function

6.17.5.1 Schema

The following schema details the information contained in the **function** element, which may appear as an element inside the **fileSet** element.



6.17.5.2 Description

A **function** specifies information about a software function. **function** contains the **replicate** attribute (optional; type: **boolean**; default: **false**); when set to **true**, the generator compiles a separate object module for each instance of the component in the design. This allows the function to be called with different attributes for each instance within the design (e.g., base address). **function** has the following elements:

- entryPoint** (optional; type: *Name*) is the entry point name for the function or subroutine.
- fileRef** (mandatory; type: *token*) reference to the file that contains the entry point for the function. The value of this element shall match an attribute in **file/fileId**. See [6.17.2](#).
- returnType** (optional) is an enumerated *string* type that indicates the return type for the function. The two possible values are **int** and **void**.
- argument** (optional; type: *nameValuePairType*, see [C.20](#)) specifies the arguments passed to the **function** when making a call. All arguments shall be passed in the order presented in this

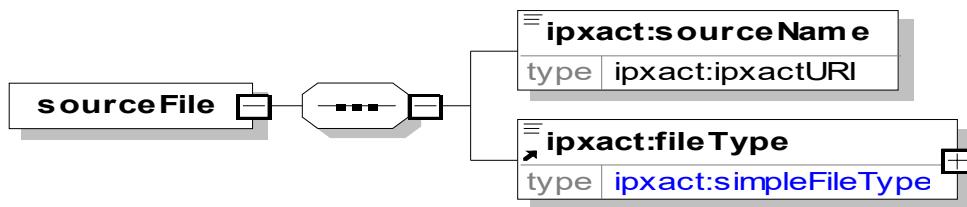
description. The **dataType** (mandatory) attribute specifies the type for this argument, e.g., an `int` or `boolean`. The **argument** element also allows for vendor attributes to be applied.

- e) **disabled** (optional; type: *unsignedBitExpression*, see [C.7.9](#)) disables the software function. When **1**, the software function is not available for use. When **0**, the function is available. If this element is not present, its effective value is **0**.
- f) **sourceFile** (optional) references any source files. The order of the source files may be important, as this could indicate a compile order. See [6.17.6](#).

6.17.6 sourceFile

6.17.6.1 Schema

The following schema details the information contained in the **sourceFile** element, which may appear as an element inside the **function** element.



6.17.6.2 Description

The **sourceFile** element specifies the location of the source files for this **function**. All source files shall be processed in the order presented in this description.

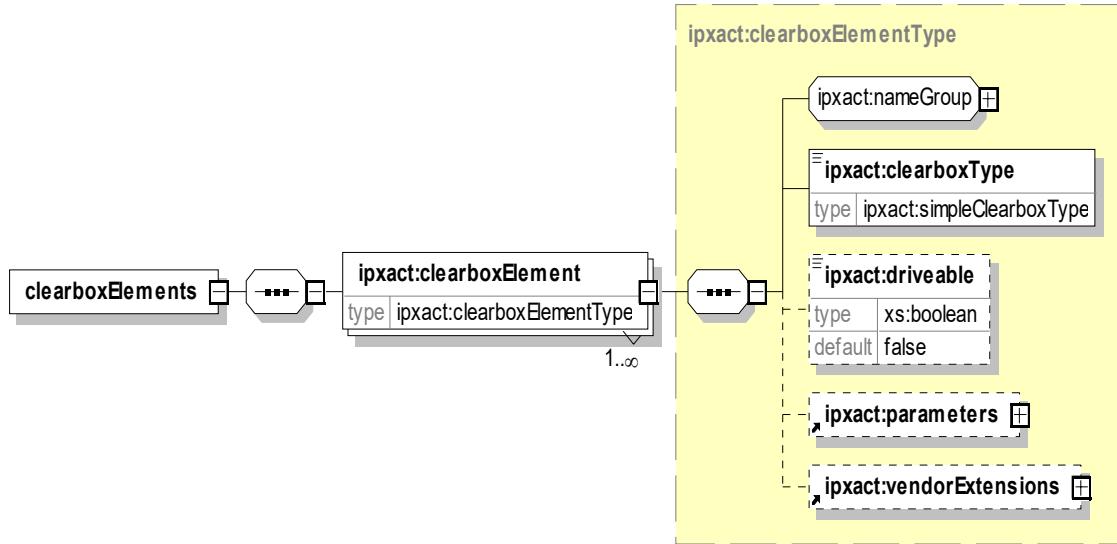
- a) **sourceName** (mandatory; type: *ipxactURI*, see [D.11](#)) contains an absolute or relative (to the location of the containing document) path to a file name or directory. The path may also contain environment variables from the host system, used to abstract the location of files.
- b) **fileType** (mandatory) group contains one or more of the elements defined in [C.16](#). If the **fileType** is set to **other**, the optional attribute **other** can be set to indicate a user-defined file type. No semantic meaning is inferred from the user-defined file type.

6.18 Clear box elements

Verification IP (VIP), with monitor bus interfaces, connect to an active bus interface to monitor only that interface's protocol for a variety of uses. Other verification tools may require access to component IP in a design, at a level deeper than the interfaces defined for the component. A clear box element provides such access. This can be used in situations where internal registers, pins, signals, or whole IP-XACT interfaces need to be monitored or driven by VIP.

6.18.1 Schema

The following schema details the information contained in the **clearboxElements** element, which may appear as an element inside the top-level **component** element.



6.18.2 Description

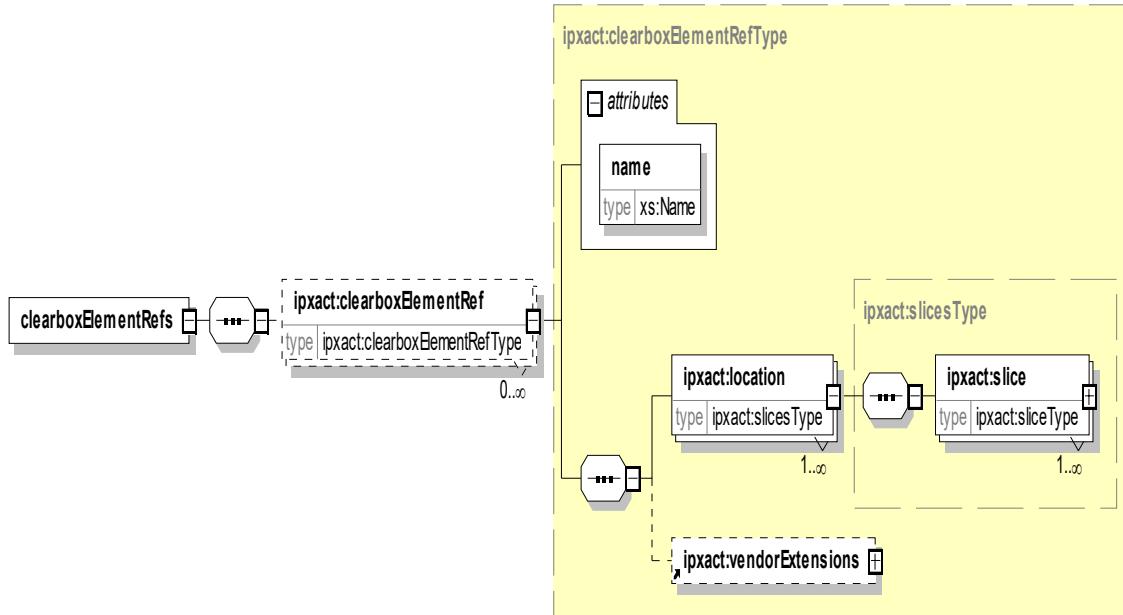
The **clearboxElements** element contains a list of one or more **clearboxElement** elements. Each **clearboxElement** element contains the following elements:

- nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **clearboxElements** element.
- clearboxType** (mandatory) documents this clear box element's referent: a **pin**, **signal**, or **interface** within the component. **pin** indicates a port on an internal instance in this component can be mapped to physical signals. **signal** indicates a signal between two internal instances in this component can be mapped to physical signals. **interface** indicates a group of signals that can be addressed as a single name.
 In each case, the view-specific path is contained in the matching **model/instantiations/componentInstantiation/clearboxElementRefs** element.
- driveable** (optional; type: **boolean**; default: **false**), when **true**, indicates the clear box describes a point within the IP that can be driven, i.e., forced to a new value. If **false**, the clear box references a point that cannot be driven. If this element is not present, its effective value is **false**.
- parameters** (optional) specifies any parameter names and types for a clear box that can be parameterized. See [C.21](#).
- vendorExtensions** (optional) provides a space for any vendor-specific extensions. See [C.27](#).

6.19 Clear box element reference

6.19.1 Schema

The following schema details the information contained in the **clearboxElementRefs** element, which may appear as an element inside the **component/model/instantiations/component** element.



6.19.2 Description

The **clearboxElementRefs** element contains a list of one or more **clearboxElementRef** elements. The **clearboxElementRef** makes a reference to a **clearboxElement** of the component and defines the view-specific path to the element. The **name** (mandatory; type: *Name*) attribute identifies the **clearboxElement** in the containing component to which the following **location** applies. **clearboxElement** element contains the following elements:

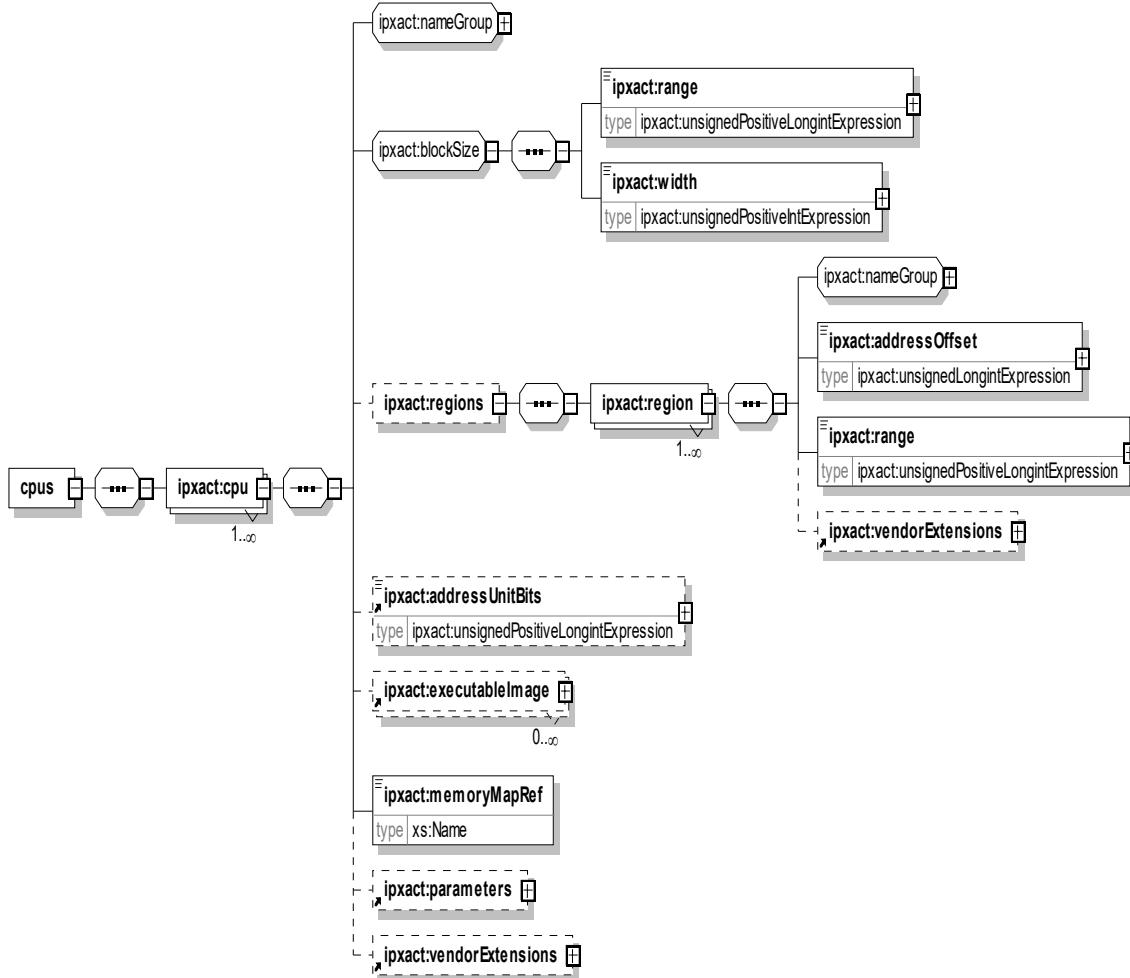
- location** (mandatory) specifies the HDL-specific path(s) to be associated with the indicated **clearboxElement**. A **location** element defines a HDL path to the HDL representation of the IP-XACT object. If there are multiple **location** elements, the IP-XACT object has several copies in the HDL. Each **location** element contains one or more **slice** elements. See [C.1.4](#).
- vendorExtensions** (optional) provides a space for any vendor-specific extensions. See [C.27](#).

See also [SCR 15.3](#).

6.20 CPUs

6.20.1 Schema

The following schema details the information contained in the **CPUs** element, which may appear as an element inside the top-level **component** element.



6.20.2 Description

The **cpus** element contains an unbounded list of **cpu** elements for the containing component. The **cpu** element describes a containing component with a programmable core that has some addressable space described by a range and width. That space may be divided into regions. The reference memory map is mapped into the addressable space. It may contain subspace maps that reference initiator interface to create a global memory map for the programmable core in the same way as done for target interfaces. See [Clause 13](#).

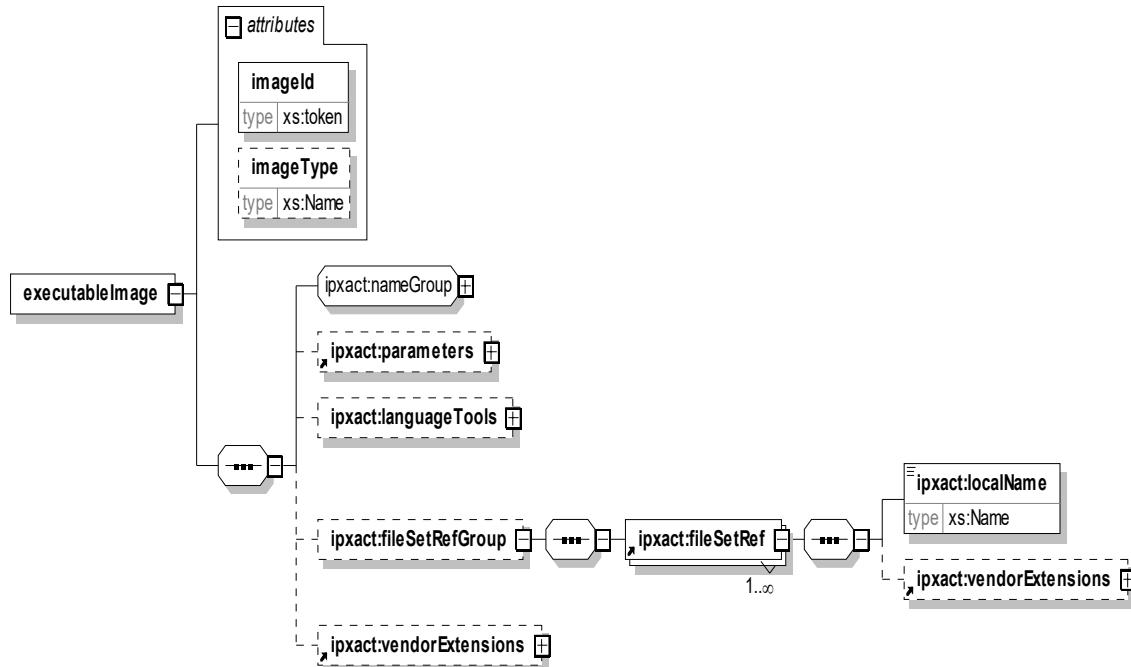
- nameGroup** group is defined in [C.19](#). The **name** element shall be unique within the containing **component** element.
- blockSize** group includes the following (see [6.11.1](#)):
 - range** (mandatory; type: **unsignedPositiveLongintExpression**, see [C.7.14](#)) gives the address range of a **cpu** element. See [C.7.14](#).

- 2) **width** (mandatory) is the bit width of a row in a **cpu** element. The type of this element is set to **nonNegativeInteger**. The width element is of type **unsignedIntExpression**. See [C.7.11](#).
- c) **regions** contains an unbounded list of **region** elements. Each **region** describes the location and size of an area in the containing **CPUs**. The **regions** element contains the following elements:
 - 1) **nameGroup** group is defined in [C.19](#). The region name shall be unique within the containing **regions** element.
 - 2) **addressOffset** (mandatory; type: **unsignedLongintExpression**, see [C.7.12](#)) describes, in addressing units from the containing element's **addressUnitBits** element, the offset from the start of the **cpu** element.
 - 3) **range** (mandatory; type: **unsignedPositiveLongintExpression**, see [C.7.14](#)) gives the address range of a region. This is expressed in terms of the number of addressable units of the **cpu** element. The size of an addressable unit is defined inside the **addressUnitBits** element.
 - 4) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this address space. See [C.27](#).
- d) **addressUnitBits** (optional; type: **unsignedPositiveLongintExpression**, see [C.7.14](#)) defines the number of data bits in each address increment of the **cpu** element. If this element is not present, it is presumed to be 8.
- e) **executableImage** (optional) describes the details of an executable image that can be loaded and executed in this **cpu**.
- f) **memoryMapRef** (mandatory) references a name of a memory map defined in the containing description. The memory map contains information about the range of registers, memory, or other address blocks. See [6.12](#).
- g) **parameters** (optional) specifies any **cpu**-type parameters. See [C.21](#).
- h) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **cpu**. See [C.27](#).

6.20.3 executableImage

6.20.3.1 Schema

The following schema details the information contained in the **executableImage** element, which may appear inside a **cpu** element.



6.20.3.2 Description

The **executableImage** element describes the details of an executable image that can be loaded and executed on a **cpu** and contains the following attributes and elements:

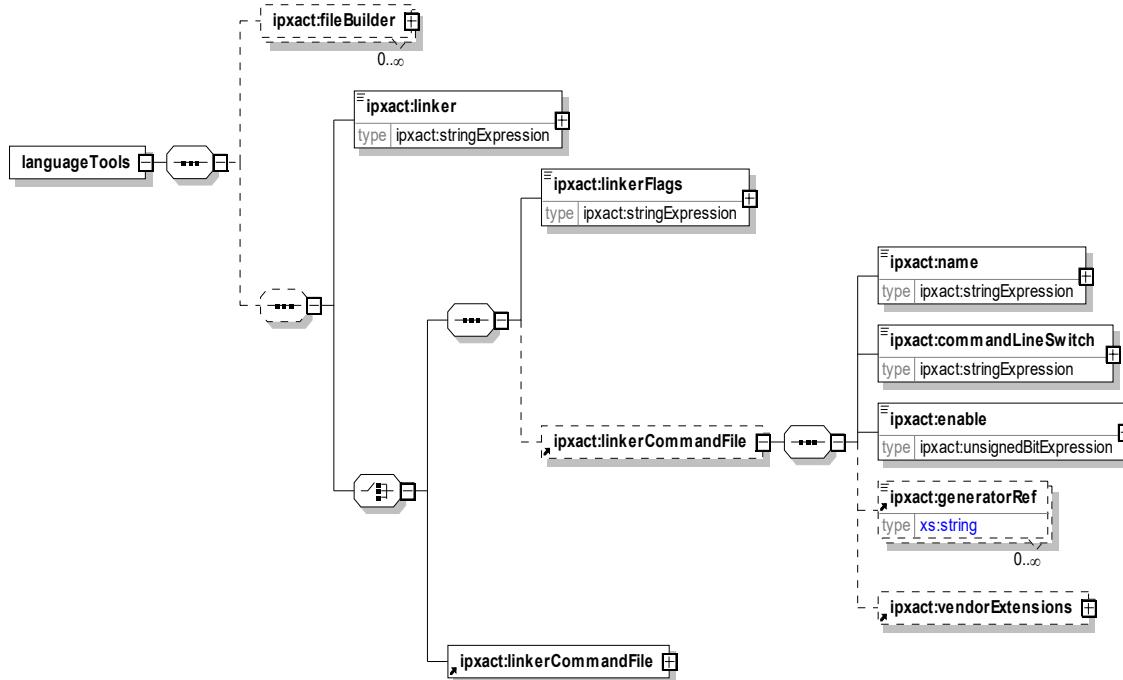
- imageId** (mandatory; type *token*) attribute uniquely identifies the **executableImage** for reference in a **fileSet/function/fileRef**.
- imageType** (optional; type *Name*) attribute can describe the binary executable format (e.g., raw binary). The list of possible values is user-defined.
- nameGroup** group is defined in [C.19](#). The **executableImage** name shall be unique within the containing **cpu** element.
- parameters** (optional) specifies any parameter data value(s) for this executable object. See [C.21](#).
- languageTools** (optional) contains further elements to describe the information needed to build the executable image. See [6.11.3](#).
- fileSetRefGroup** (optional) element contains a list of **fileSetRef** subelements, each one containing the name of a file set associated with this **executableImage**. See [6.17](#).
- vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this address space. See [C.27](#).

See also [SCR 9.1](#), [SCR 9.6](#), and [SCR 9.7](#).

6.20.4 languageTools

6.20.4.1 Schema

The following schema details the information contained in the **languageTools** element, which may appear as an element inside the **executableImage** element.



6.20.4.2 Description

The **languageTools** element contains the following list of optional elements to document a set of software tools used to create an executable binary documented by the parent **executableImage** element. Multiple **languageTools** information can be created to reflect various software tool sets that can create this executable binary file.

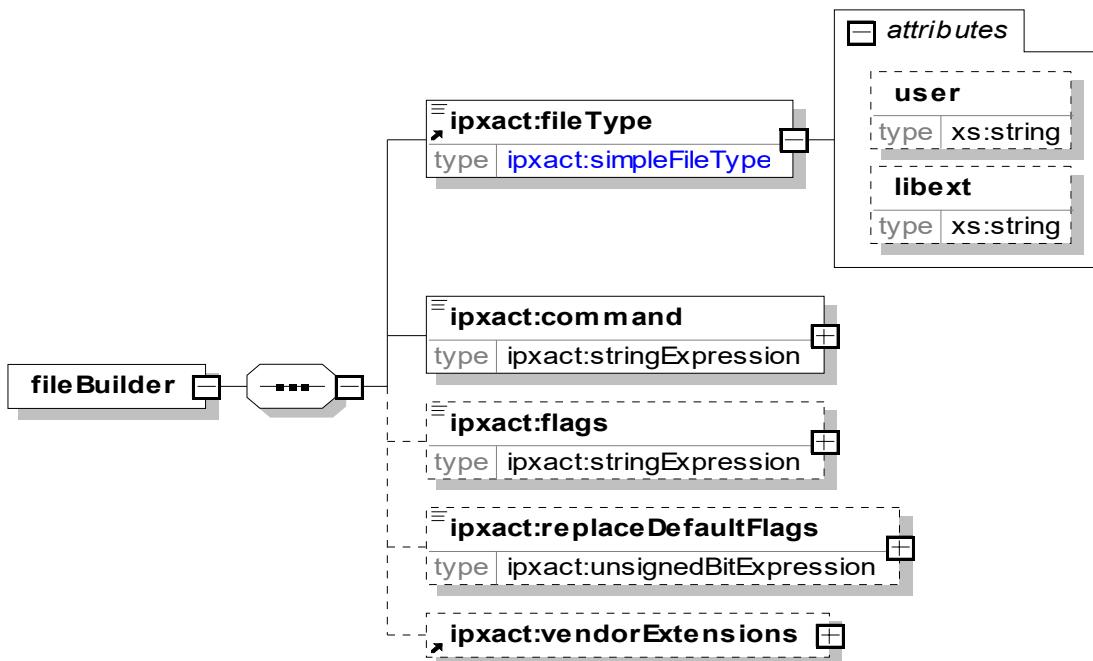
- fileBuilder** (optional) contains the information details of a compiler or assembler for software source code. See [6.20.5](#).
- linker** (optional; type: *stringExpression*, see [C.7.5](#)) documents the link editor associated with the software tools described in **fileBuilder**.
- A **languageTools** also contains one of the following—a) a **linkerFlags** element and optionally a **linkerCommandFile**; or b) a **linkerCommandFile**.
 - linkerFlags** (optional; type: *stringExpression*, see [C.7.5](#)) can also be associated with any **linker** information.
 - linkerCommandFile** (optional) documents a file containing commands the linker follows. The **linkerCommandFile** element contains information related to contents of the **linker** and **linkerFlags** elements, specifically about a file containing linker commands. It contains the following subelements:
 - name** (mandatory; type: *ipxactURI*, see [D.11](#)) documents the location and name of the file containing commands for the linker.
 - commandLineSwitch** (mandatory type: *stringExpression*, see [C.7.5](#)) documents the flag on the command line specifying the linker command file.

- iii) **enable** (mandatory; type: *unsignedBitExpression*, see [C.7.9](#)) indicates whether to use this linker command file in the default scenario. The following also apply:
 - enable** is **1** with a **generatorRef**: run the generator to link the **executableImage**; it may use the other elements to link the **executableImage**.
 - enable** is **1** with no **generatorRef**: run the linker with the **commandLineSwitch** *name* (the command file).
 - enable** is **0**: run the linker with **linkerFlags**.
- iv) **generatorRef** (optional; type: *string*) references the generator (in the containing component) that creates and launches the linker command. There may be any number of these elements present. See [6.16](#).
- v) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces applicable to using this linker. See [C.27](#).

6.20.5 fileBuilder

6.20.5.1 Schema

The following schema details the information contained in the **fileBuilder** element, which may appear as an element inside a **languageTools** element within the **executableImage** element.



6.20.5.2 Description

The **fileBuilder** element contains the following elements:

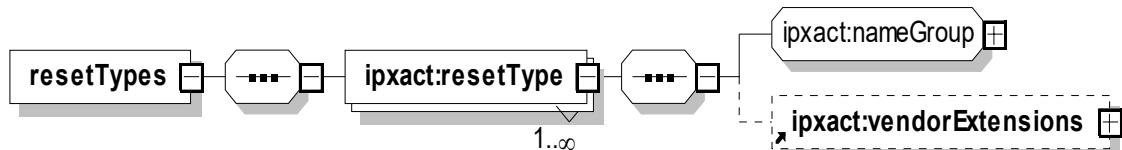
- a) **fileType** (mandatory) indicates the type of file referenced by IP-XACT. If the required file type is not available, set the **fileType** to **user** and then set the **user** attribute to the desired file type. Any string value is valid in the **user** attribute, but the file may not be able to be processed as that specific type by a DE.
- b) **command** (mandatory; type: *stringExpression*, see [C.7.5](#)) defines a compiler or assembler tool that processes the software of this type.

- c) **flags** (optional; type: *stringExpression*, see [C.7.5](#)) documents any flags to be passed along with the software tool command.
- d) **replaceDefaultFlags** (optional; type: *unsignedBitExpression*, see [C.7.9](#)) documents, when **1**, flags that replace any of the default flags from a build script generator. If **0**, the flags contained in the flags element are appended to the current command. If the value is **1** and the **flags** element is empty or does not exist, this has the effect of clearing all the flags in build script generator. If this element is not present, its effective value is **0**.
- e) **vendorExtensions** (optional) holds vendor-specific data from other namespaces applicable to building this software source code file into an executable object file. See [C.27](#).

6.21 Reset types

6.21.1 Schema

The following schema details the information contained in the **resetTypes** element, which may appear as an element inside the top-level **component** element.



6.21.2 Description

The **resetTypes** element contains an unbounded list of **resetType** elements for the containing component. The **resetType** element describes any user-defined reset types referenced within field reset elements. The reset type named **HARD** is the default and should be treated as being pre-defined. It is not legal to explicitly define a **resetType** element using that name. Register fields that do not reference a reset type are presumed to be of type **HARD** reset. A **resetType** contains the following elements:

- a) **nameGroup** group is defined in [C.19](#). The **resetType** name shall be unique within the containing **resetTypes** element.
- b) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **resetType**. See [C.27](#).

See also [SCR 7.19](#).

7. Design descriptions

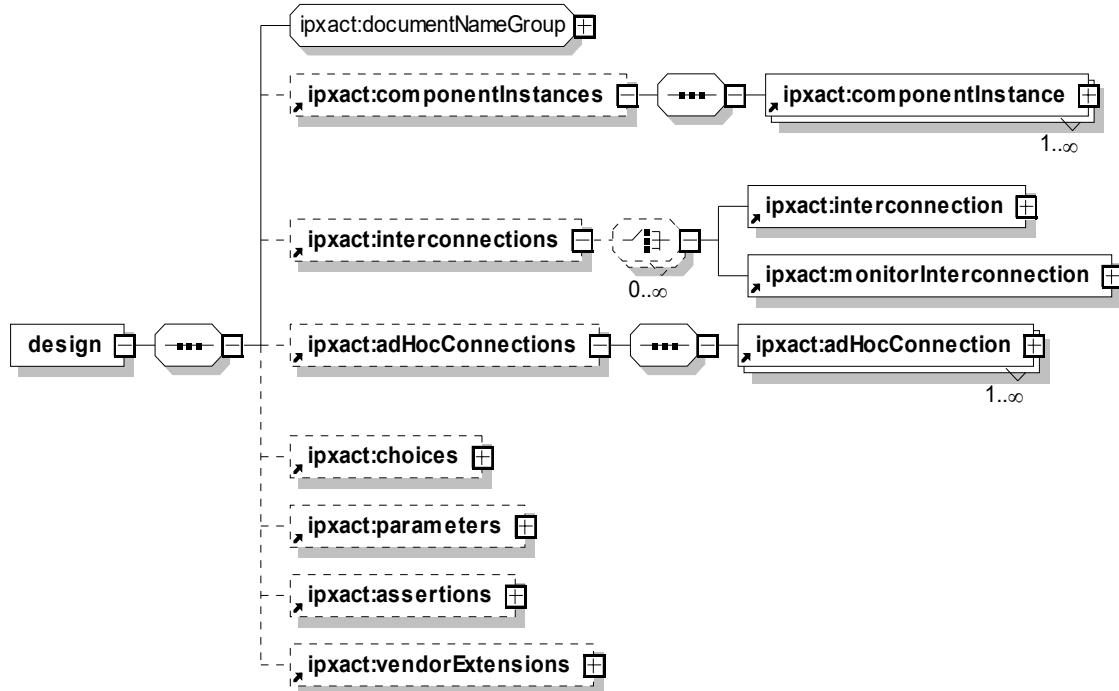
7.1 Design

An IP-XACT *design* is the central placeholder for the assembly of component objects meta-data. A design describes a list of components referenced by this description, their configuration, and their interconnections to each other. The interconnections may be between interfaces or between ports on a component. A design description is analogous to a schematic of components.

While a design description, with referenced components and interconnections, describes most of the information for a design, some information is missing, such as the exact port names used by a bus interface. To resolve this a component description (referred to as a *hierarchical component*) is used. This *component description* contains a view with a reference to the design description. Together, the component and referenced design description form a complete single-level hierarchical description. From this point, it is simple to create additional hierarchical descriptions by including hierarchical component description in design descriptions.

7.1.1 Schema

The following schema details the information contained in the **design** element, which is one of the top-level elements of the schema.



7.1.2 Description

The **design** element describes a list of referenced components, their configuration, and interconnections to each other. Each element of a **design** is detailed in the rest of this clause; the main sections of a **design** are as follows:

- a) **documentNameGroup** describes a **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. See [C.11](#). The **versionedIdentifier** or VLVN consists of four subelements for a top-level IP-XACT element. See [C.28](#).
- b) **componentInstances** (optional) contains an unbounded list of **componentInstance** elements, documenting components that are instantiated (referenced) inside the design (see [7.2](#)).
- c) **interconnections** (optional) contains the connections between bus interfaces of components listed inside the design. The **interconnection** element is used to document active and hierarchical interface connections, and the **monitorInterconnection** element is used to document a connection between an active interface and a monitor interface (see [7.3](#)).
- d) **adHocConnections** (optional) contains an unbounded list of **adHocConnection** elements, documenting connections between component ports listed inside this design (see [7.5](#)).
- e) **choices** (optional) specifies multiple enumerated lists. These lists are referenced by other sections of this design description. See [C.8](#).
- f) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this **design**. See [C.21](#).
- g) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.5](#).
- h) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).

See also [SCR 1.10](#).

7.2 Design component instances

7.2.1 Schema

The following schema details the information contained in the **componentInstances** element, which may appear as an element inside the top-level **design** element.



7.2.2 Description

The **componentInstance** element documents the existence of a component in a design. This element contains the following subelements:

- instanceName** (mandatory; type: *Name*) assigns a unique name for this instance of the component in this design. The value of this element shall be unique inside a **design** element.
- displayName** (optional; type: *string*) allows a short descriptive text to be associated with the containing element. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- shortDescription** (optional; type: *string*) is a more compact textual description of the containing element compared to description. See [C.25](#).
- description** (optional; type: *string*) allows a textual description of the instance.
- componentRef** (mandatory; type *configurableLibraryRefType*, see [C.10](#)) is a reference to a component description (see [6.1](#)) for this component instance. The **componentRef** element contains four attributes to specify a unique VLN and an optional **configurableElementValues** element, which is used to document values for parameters to be passed into the referenced document.

- f) **powerDomainLinks** (optional) contains one or more **powerDomainLink** elements. Each powerDomainLink has a **externalPowerDomainReference** (mandatory; type *stringExpression*) element and a **internalPowerDomainReference** (mandatory; type *Name*) element.
 - 1) The **externalPowerDomainReference** value shall evaluate to a powerDomain defined in the component referencing this design.
 - 2) The **internalPowerDomainReference** shall reference a powerDomain defined in the component referenced by this componentInstance.
 - g) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).

See also [SCR 1.9](#), [SCR 1.33](#), [SCR 1.34](#), [SCR 5.3](#), and [SCR 5.12](#).

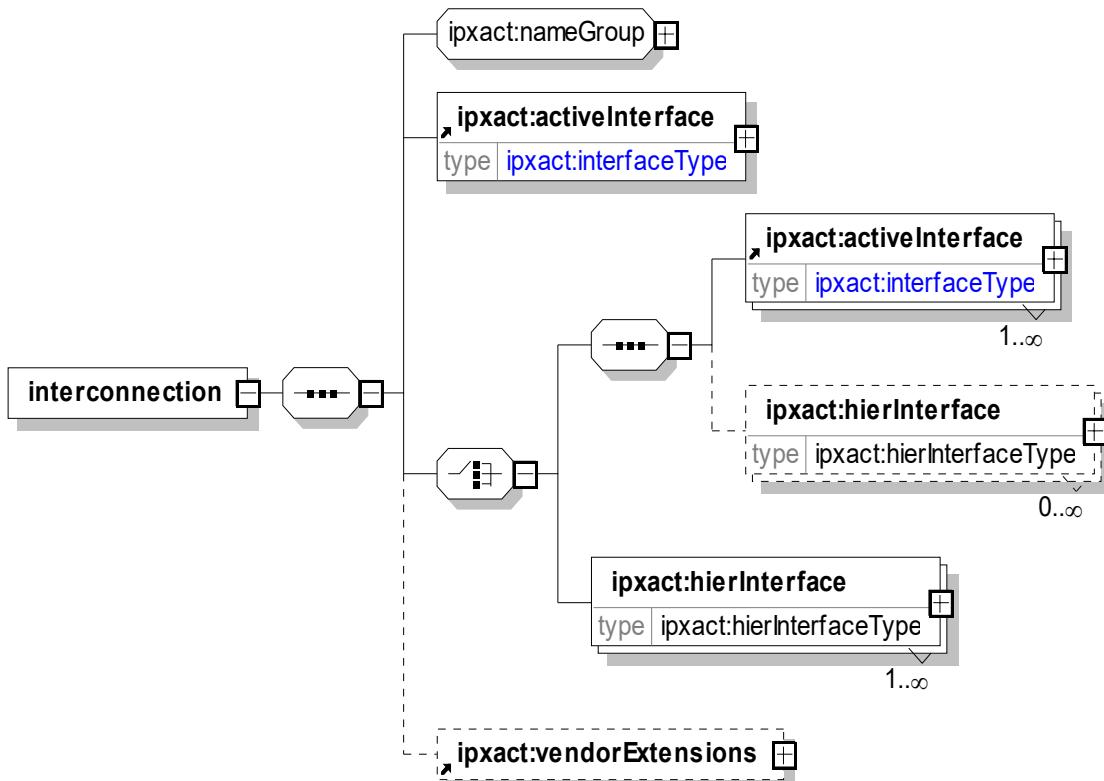
7.3 Design interconnections

The **interconnections** element contains an unbounded list of **interconnection** (see [7.3.1](#)) and **monitorInterconnection** (see [7.3.2](#)) elements. The **interconnections** element may appear as an element inside the top-level **design** element. For further description on interface connections, see [6.5.5](#).

7.3.1 interconnection

7.3.1.1 Schema

The following schema details the information contained in the **interconnection** element, which may appear as an element inside the **interconnections** element.



7.3.1.2 Description

The **interconnection** element specifies a connection between one bus interface of a component and another bus interface of a component. Each interconnection contain the following elements:

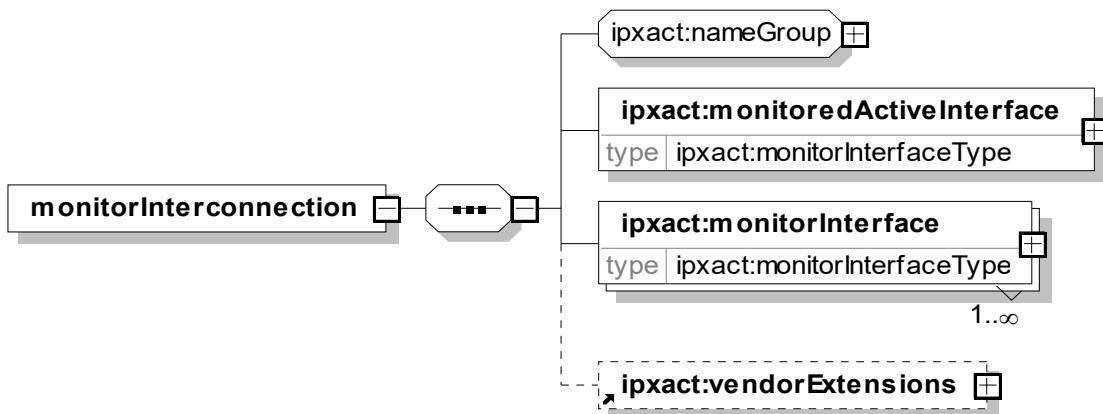
- nameGroup** group is defined in [C.19](#). The **name** elements shall be unique within the containing **interconnections** element.
- At least two interfaces need to be defined. The following combinations are possible here: one **activeInterface** element and a choice of either one or more additional **activeInterface** elements plus zero or more **hierInterface** elements, or one or more **hierInterface** elements.
 - activeInterface** (type: *interfaceType*, see [7.4](#)) specifies a non-monitor interface connection point on a component. Having more than two **activeInterface/hierInterface** elements present is allowed for broadcast connections. See [6.5.4](#).
 - hierInterface** (type: *hierInterfaceType*, see [7.4](#)) specifies a non-monitor interface connection point on the boundary of the containing component. Having more than two **activeInterface/hierInterface** elements present is allowed for broadcast connections. See [6.5.4](#).
- vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).

See also SCRs in [Table B.2](#) and [Table B.11](#).

7.3.2 monitorInterconnection

7.3.2.1 Schema

The following schema details the information contained in the **monitorInterconnection** element, which may appear as an element inside the **interconnections** element.



7.3.2.2 Description

The **monitorInterconnection** element specifies the connection between a monitored active interface on a component and a list of monitor interfaces on component instances.

- nameGroup** group is defined in [C.19](#). The **name** elements shall be unique within the containing **interconnections** element.
- monitoredActiveInterface** (mandatory; type: *moniterInterfaceType*, see [7.4](#)) specifies the component bus interface to monitor.

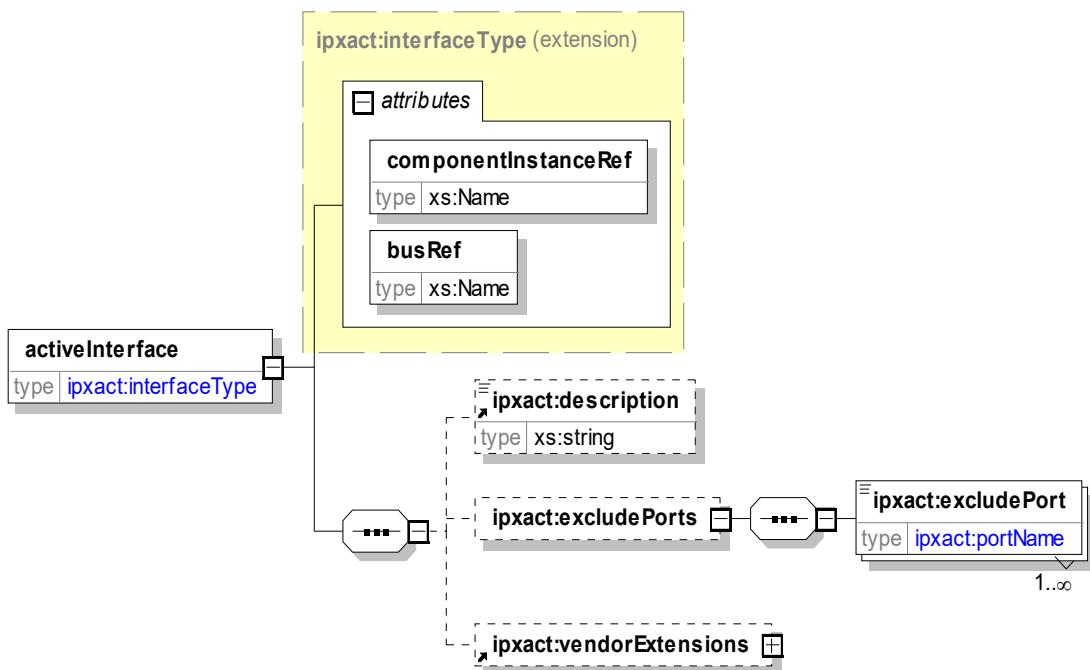
- c) **monitorInterface** (mandatory; type: *monitorInterfaceType*, see [7.4](#)) specifies the component bus interface that will do the monitoring. There may be one or more **monitorInterface** elements specified.
- d) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).

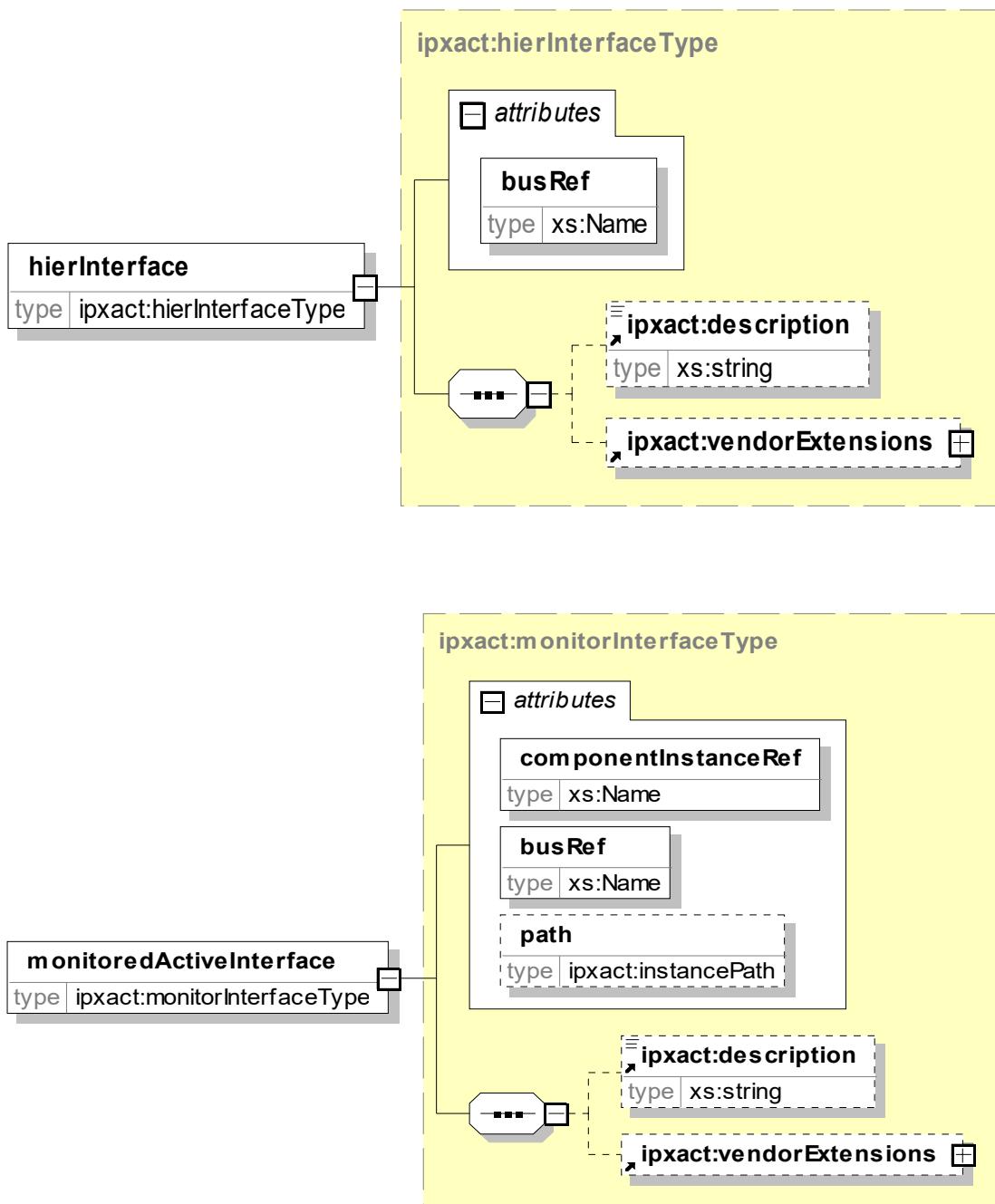
See also [SCR 6.9](#), and [SCR 6.10](#), and the SCRs in [Table B.2](#) and [Table B.4](#).

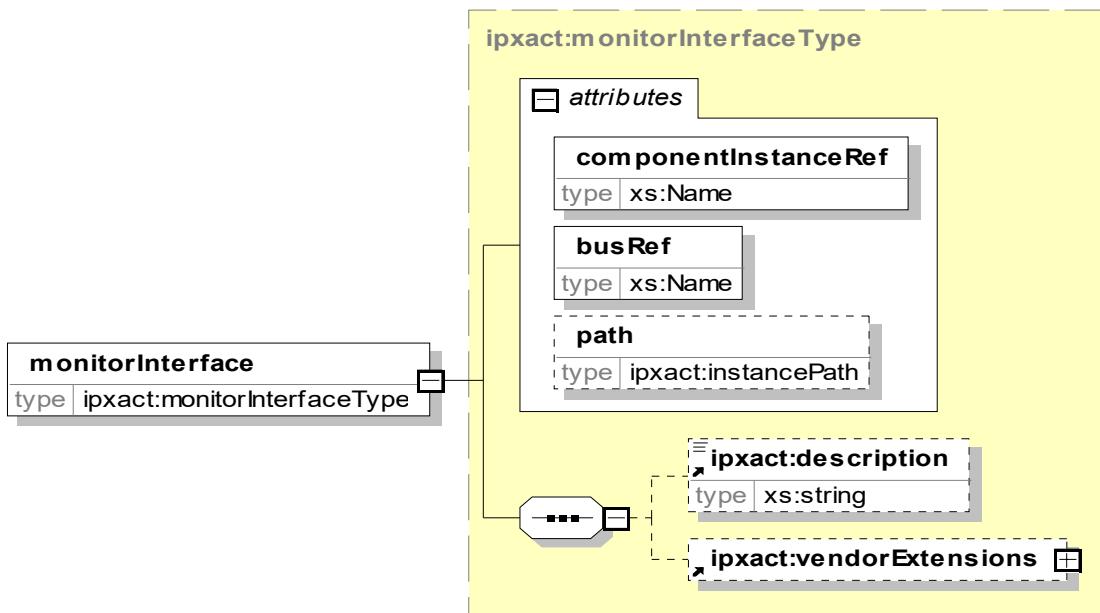
7.4 Active, hierarchical, monitored, and monitor interfaces

7.4.1 Schema

The following schemas define the information contained in the **activeInterface** element, the **hierInterface** element, the **monitoredActiveInterface** element, and the **monitorInterface** element, which may appear as elements inside the **interconnection** or **monitorInterconnection** element within the **interconnections** element.







7.4.2 Description

The **activeInterface**, **hierInterface**, **monitoredActiveInterface**, or **monitorInterface** elements specify the bus interface of a design component instance that is part of an interconnection or a monitor interconnection. They all have the following attributes:

- **busRef** (mandatory; type: *Name*) references one of the component bus interfaces. This specific bus interface needs to exist on the specified component instance. See [6.7](#).
- **description** (optional; type: *string*) allows a textual description of the instance.
- **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).

The **activeInterface**, **monitoredActiveInterface**, and **monitorInterface** elements have the following attributes:

- **componentInstanceRef** (mandatory; type: *Name*) references the instance name of a component present in the design if the path attribute is not present. This component instance name needs to exist in the specified design. See [6.1](#).

The **activeInterface** element also has the following element:

- **excludePort** (optional) contains a list of **excludePort** elements, each of which defines a physical port that should be excluded from interface driven physical connections. In effect, ports listed here are treated as if they are not defined in the **portMap** of the referenced interface.
 Names listed need to exist in a **portMap** of the referenced interface.

The **monitoredActiveInterface** and **monitorInterface** elements have the following attribute:

- **path** (optional; type: *instancePath*) defines the hierarchical path of instance names to the design that contains the component instance specified in the **componentInstanceRef** attribute. The path is a slash (/) separated list of instance names. If the **path** attribute is not present, the component referenced by **componentInstanceRef** needs to exist in the current design. See [D.4](#).

See also [SCR 2.1](#), [SCR 2.16](#), [SCR 2.18](#), [SCR 4.1](#), and [SCR 4.2](#).

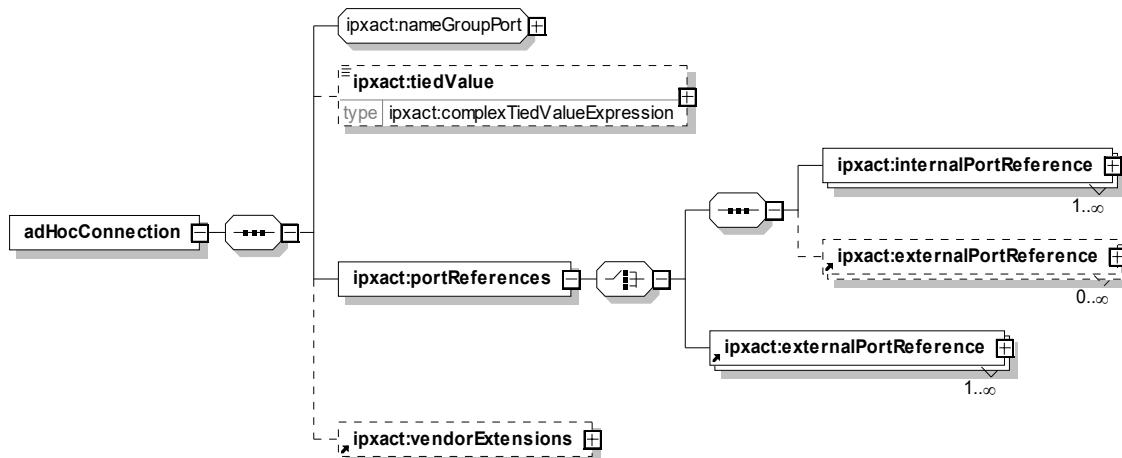
7.5 Design ad hoc connections

The name *ad hoc* is used for connections that are made on a port-by-port basis and not done through the higher-level bus interface. The same **ports** that make up a **busInterface** can be used in ad hoc connections.

IP-XACT supports two cases of ad hoc connections: the wire/structured connection (between ports having a wire/structured style) and the transactional connection (between ports having a transactional style). The direct connection between a wire-style/structured-style port and a transactional-style port is not allowed; a specific adapter component needs to be inserted in between them.

7.5.1 Schema

The following schema details the information contained in the **adHocConnection** element, which may appear as an element inside the top-level **design/adHocConnections** element.



7.5.2 Description

An **adHocConnection** specifies connections between component instance ports and/or between component instance ports and ports of the encompassing component (in the case of a hierarchical component). Each **adHocConnection** shall contain at least one port reference (internal or external). The **adHocConnection** element contains the following subelements:

- a) **nameGroupPort** group is defined in [C.19.4](#). The **name** elements shall be unique within the containing **adHocConnections** element.
 - b) **tiedValue** (optional) specifies a fixed logic value for this connection. The value of this element can be an expression that evaluates to an explicit numeric value or to the string values **default** or **open**. The value **default** implies each port should be connected to the default value defined in the port declaration. See [C.31](#) to determine when this value is applied. The value **open** implies each port is being intentionally left open (or unconnected). The **tiedValue** element is of type **complexTypeTiedValueExpression**. See [C.7.1](#).
 - c) **portReferences** (mandatory) specifies a list on internal and external port references for this **adHocConnection**. At least one reference needs to be defined; the following combinations are possible:

one or more **internalPortReference** elements plus zero or more **externalPortReference** elements or one or more **externalPortReference** elements.

- 1) **internalPortReference** references the port of a component instance. See [7.6](#).
- 2) **externalPortReference** references a port of the encompassing component where this design is referred (for hierarchical ad hoc connections). See [7.6](#).
- d) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).

See also [SCR 6.25](#).

7.5.3 Ad hoc wire and structured connection

For ad hoc connections between wire-style and structured-style ports, IP-XACT states that the style can be **wire** or **structural**.

See also [SCR 6.9](#) and [SCR 6.24](#).

7.5.4 Ad hoc transactional connection

For ad hoc transactional connections, IP-XACT requires the style of each port to be **transactional**.

See also [SCR 6.10](#) and [SCR 6.22](#).

7.5.5 Interaction rules between an interface-based connection and ad hoc connections

This subclause details the rules defining the interaction between interface-based connection and ad hoc connections. These rules apply to wire ports, structured ports, and transactional ports. Here, the term *pin* is used to designate a component port, regardless of its style.

Rule A—A pin P is involved in an interface-based connection whenever it appears in the port map of the interface (linked to logical port L) and one of the following is also **true**:

- The logical port L on the opposite (connected) interface appears in the port map; therefore, there is an actual pin to connect to on the opposite side of the interface.
- The definition of the logical port L (in its abstraction definition) contains the definition of a default value. In this case, the pin will be connected to that default value.

Rule B—Connections specified via the **adHocConnection** element are always made in addition to any connections implied by interface connections.

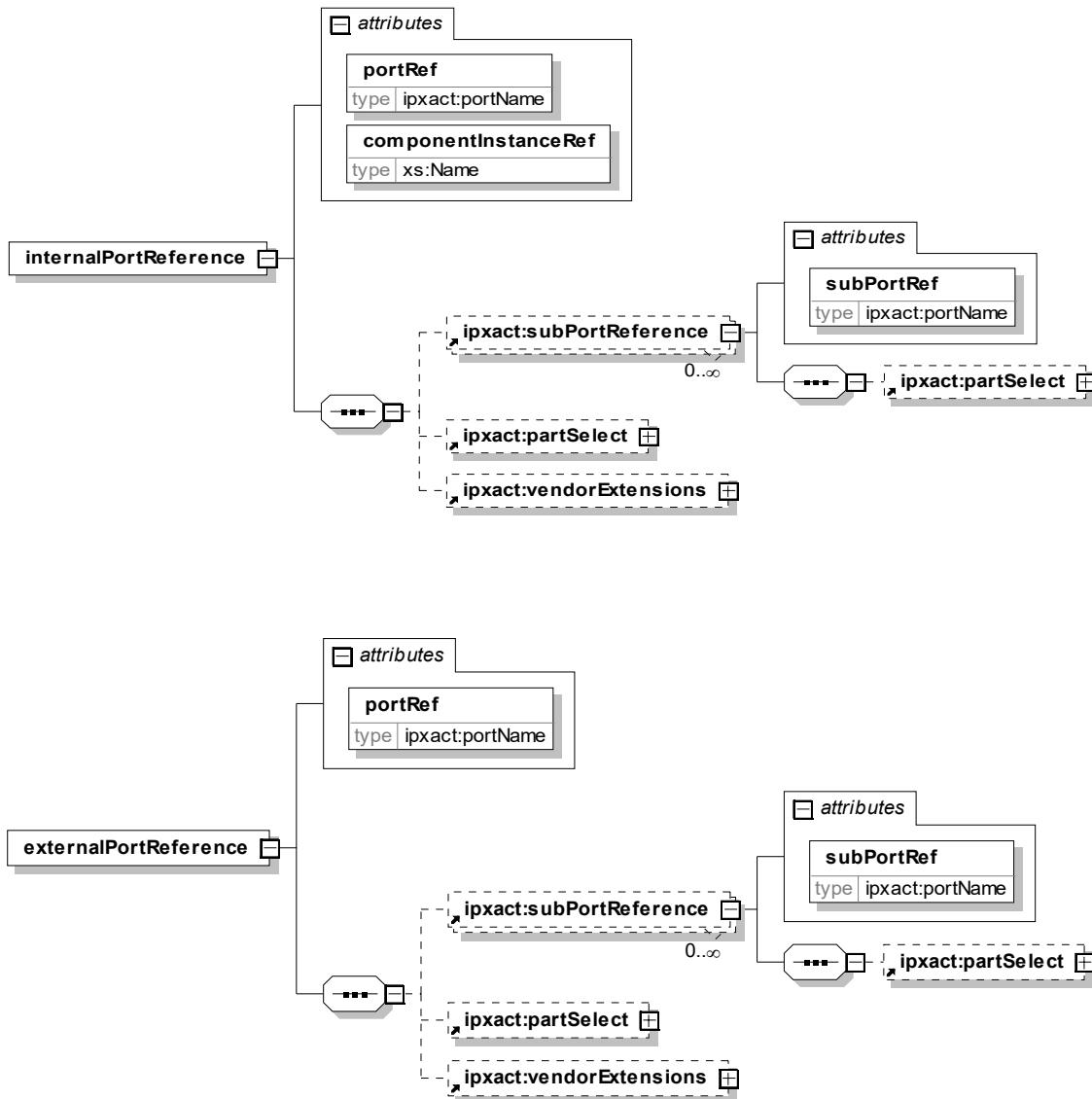
Rule C—All connections to a pin are made as the result of either an interface connection or an ad hoc connection.

Rule A documents when a pin is considered connected via an interface. *Rule B* documents when a pin is conservatively connected due to inclusion in an ad hoc connection. *Rule C* states that *Rules A* and *B* are the only ways to cause a connection to occur. In other words, a pin is not considered connected from a component to its default value unless it appears in an ad hoc connection as `tiedValue == "default"`.

7.6 Port references

7.6.1 Schema

The following schema details the information contained in the **portReferences**, which may include **internalPortReference** or **externalPortReference** elements, and which may appear as an element within the **adHocConnections** element.



7.6.2 Description

The **portReferences** element specifies the list of internal and external port references for this **adHocConnection**. This defines the full set of pins and ports involved in a single ad-hoc connection. **adHocConnection** has the following elements:

- a) **internalPortReference** references the port of a component instance. It has the following subelements:

- 1) **componentInstanceRef** (mandatory; type: *Name*) references the component instance name for the port. See [6.1](#).
 - 2) **portRef** (mandatory; type: *Name*) references the port name on the specific component instance. See [6.15.7](#).
 - 3) **subPortReference** (optional) contains an attribute subPortRef (mandatory; type *portName*) and zero or more partSelect elements. The first **subPortReference** shall reference a subPort of a structured port referenced by the **portRef** element. Each consecutive **subPortReference** shall reference a subPort of the **subPort** referenced in the previous **subPortReference** element. The range elements are used to select subelements in the referenced subPort. If the last **subPortReference** does not reference a wire port then all **subPorts** below the referenced port are assumed to be referenced recursively. See [C.25](#).
 - 4) **partSelect** (optional) defines the sub-range of the port being referenced. If no **partSelect** is specified, the entire port is assumed to be connected. See [C.22](#).
 - 5) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).
- b) **externalPortReference** references a port of the encompassing component where this design is referred (for hierarchical ad hoc connections). It has the following subelements:
- 1) **portRef** (mandatory; type: *Name*) references the port name on the encompassing component. See [6.15.7](#).
 - 2) **subPortReference** (optional) contains an attribute subPortRef (mandatory; type *portName*) and zero or more partSelect elements. The first **subPortReference** shall reference a subPort of a structured port referenced by the **portRef** element. Each consecutive **subPortReference** shall reference a subPort of the **subPort** referenced in the previous **subPortReference** element. The range elements are used to select subelements in the referenced subPort. If the last **subPortReference** does not reference a wire port then all **subPorts** below the referenced port are assumed to be referenced recursively. See [C.25](#).
 - 3) **partSelect** (optional) defines the sub-range of the port being referenced. If no **partSelect** is specified, the entire port is assumed to be connected. See [C.22](#).
 - 4) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).

See also [SCR 6.53](#), [SCR 6.54](#), and [SCR 6.55](#).

8. Abstracter descriptions

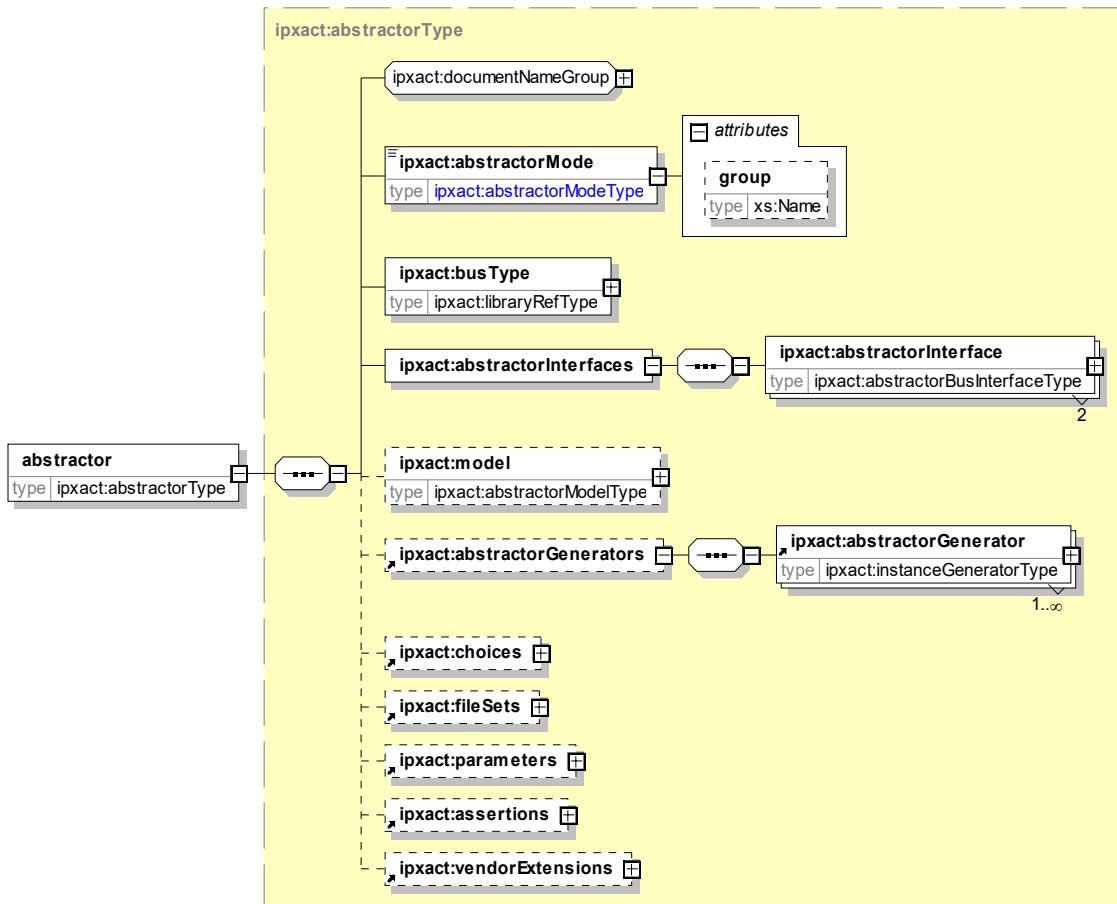
Designs that incorporate IP models using different interface modeling styles (e.g., TLM and RTL modeling styles) may contain interconnections between such component interfaces using different abstractions of the same bus type. An IP-XACT description may describe how such interconnections are to be made using a special-purpose object called an *abstracter*. An abstracter is used to connect between two different abstractions of the same bus type (e.g., an APB_RTL and an APB_TLM).²⁵ An abstracter shall contain only two interfaces, which shall be of the same bus definition and different abstraction definitions.

Unlike a component, an abstracter is not referenced from a design description, but instead is referenced from a design configuration description. See [Clause 11](#).

8.1 Abstracter

8.1.1 Schema

The following schema details the information contained in the **abstracter** element, which is one of the top-level elements in the IP-XACT specification used to describe an abstracter.



²⁵APB = AMBA® peripheral bus. AMBA is an open specification on-chip backbone for interconnecting IP blocks. AMBA is a registered trademark of Arm Limited. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of this product. Equivalent products may be used if they can be shown to lead to the same results.

8.1.2 Description

Each element of an **abstractor** is detailed in the rest of this clause; the main sections of an **abstractor** are as follows:

- a) **documentNameGroup** describes **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. The **versionedIdentifier** or VLVN consists of four subelements for a top-level IP-XACT element. See [C.28](#).
- b) **abstractorMode** (mandatory) determines the mode of the two interfaces contained in **abstractorInterfaces**. The abstractor can be inserted in a connection between two instances or between an instance and an exported interface. The **abstractorMode** element can take one of the following values:
 - 1) **initiator** specifies for
 - i) Initiator-to-mirrored-initiator connection—the first interface connects to the initiator interface, the second connects to the mirrored-initiator interface
 - ii) Exported initiator connection—the first interface connects to the initiator interface, the second connects to the exported interface
 - iii) Exported mirrored-initiator connection—the first interface connects to the exported interface, the second connects to the mirrored-initiator interface
 - 4) **target** specifies for
 - i) Mirrored-target-to-target connection—the first interface connects to the mirrored-target interface, the second connects to the target interface
 - ii) Exported target connection—the first interface connects to the exported interface, the second connects to the target interface
 - iii) Exported mirrored-target connection—the first interface connects to the mirrored-target interface, the second connects to the exported interface
 - 4) **direct** specifies the first interface connects to the initiator interface, the second connects to the target interface. This option is not allowed for an exported interface.
 - 5) **system** specifies for
 - i) System-to-mirrored-system connection—the first interface connects to the system interface, the second connects to the mirrored-system interface
 - ii) Exported system connection—the first interface connects to the system interface, the second connects to the exported interface
 - iii) Exported mirrored-system connection—the first interface connects to the exported interface, the second connects to the mirrored-system interface
- c) The **group** (mandatory, when **abstractorMode** is **system**; type **Name**) attribute defines the name of the group to which this system interface belongs. The value of this **group** shall be unique inside the **abstractor** element and already defined in the referenced abstraction definition. A connection between a **system** and **mirroredSystem** interfaces shall have matching group names.
- d) **abstractorInterfaces** (mandatory) are interfaces that have the same bus type, but differing abstraction types. See [8.2](#).
- e) **model** (optional) specifies all the different views, ports, and model configuration parameters of the abstractor. See [8.3](#).
- f) **abstractorGenerators** (optional) specifies a list of generator programs attached to this abstractor. See [8.9](#).

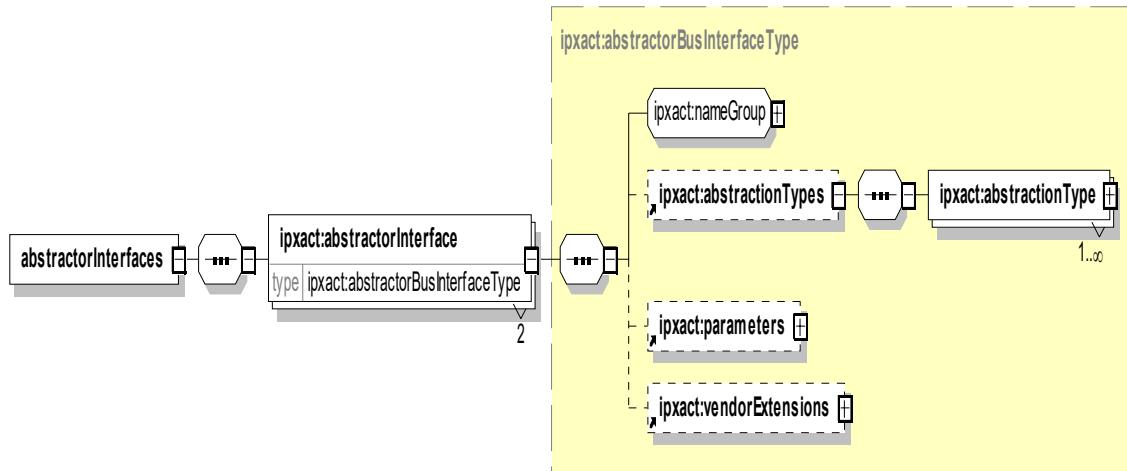
- g) **choices** (optional) specifies multiple enumerated lists, which are referenced by other sections of this abstractor description. See [C.8](#).
- h) **fileSets** (optional) specifies groups of files and possibly their function for reference by other sections of this abstractor description. See [6.17](#).
- i) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this abstractor. See [C.21](#).
- j) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.5](#).
- k) **vendorExtensions** (optional) contains any extra vendor-specific data related to the abstractor. See [C.27](#).

See also [SCR 1.10](#), [SCR 1.11](#), and [SCR 3.15](#).

8.2 Abstractor interfaces

8.2.1 Schema

The following schema defines the information contained in the **abstractorInterfaces** element, which appears within an **abstractor** description.



8.2.2 Description

The **abstractorInterfaces** element contains a list of two **abstractorInterface** elements. Each **abstractorInterface** element defines properties of this specific interface in an abstractor. The **abstractorInterface** element also allows for vendor attributes to be applied. Each **abstractorInterface** contains the following elements:

- a) **nameGroup** group is defined in [C.19](#). The **abstractorInterface** name shall be unique within the containing **abstractorInterfaces** element.
- b) **abstractionTypes** (optional) specifies the abstraction definition where this bus interface is referenced. An abstraction definition (the **abstractionType** element; see [6.7.7.2b](#)) describes the low-level attributes of a bus description (see [5.3](#)).
- c) **parameters** (optional) specifies any parameter data value(s) for this bus interface. See [C.21](#).
- d) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces that is applicable to this bus interface. See [C.27](#).

8.3 Abstracter models

8.3.1 Schema

The following schema defines the information contained in the abstracter **model** element, which may appear within an **abstracter** description.



8.3.2 Description

The **model** element describes the views, ports, and model related parameters of an abstracter. A **model** element may contain the following:

- views** (optional) contains a list of all the views for this object. An object may have many different views. An RTL view may describe the source hardware module/entity with its pin interface; a software view may define the source device driver C file with its .h interface; a documentation view may define the written specification of this IP. See [8.4](#).

- b) **instantiations** (optional) specifies the component instantiations for all views (as an **instantiationsGroup**). See [6.15.2](#).
- c) **ports** (optional) contains the list of ports for this object. A ports is an external connection from the object. An object may have only one set of ports that shall be valid for all views. See [8.5](#).

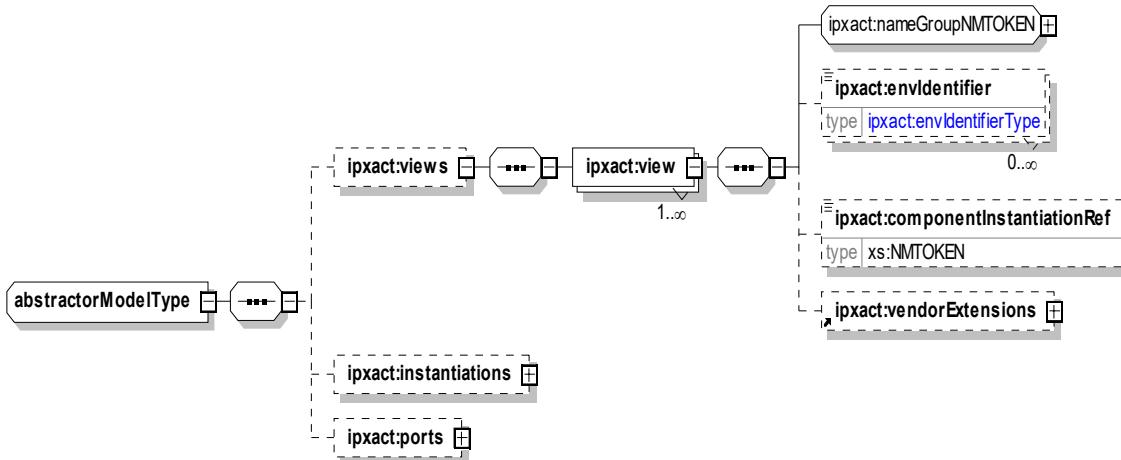
8.4 Abstractor views

8.4.1 Schema

The following schema defines the information contained in the **views** element, which appears within the **model** element of an **abstractor** description.

This schema is almost identical to the **component/model/views/view** element (see [6.15.1](#)), except:

- Abstractors have no **designInstantiationRef** elements.
- Abstractors have no **designConfigurationInstantiationRef** elements.



8.4.2 Description

A **views** element describes an unbounded set of **view** elements. Each **view** element specifies a representation level of an abstractor. It contains the following elements:

- a) **nameGroupNMTOKEN** group is defined in [C.19.5](#). The **name** elements shall be unique within the containing **views** element.
- b) **envIdentifier** (optional) designates and qualifies information about how this model view is deployed in a particular tool environment. See [6.15.1.2](#).
- c) **componentInstantiationRef** (optional) specifies a component's configuration instantiation for a particular instantiation. See [6.15.3.2](#).
- d) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See [C.27](#).

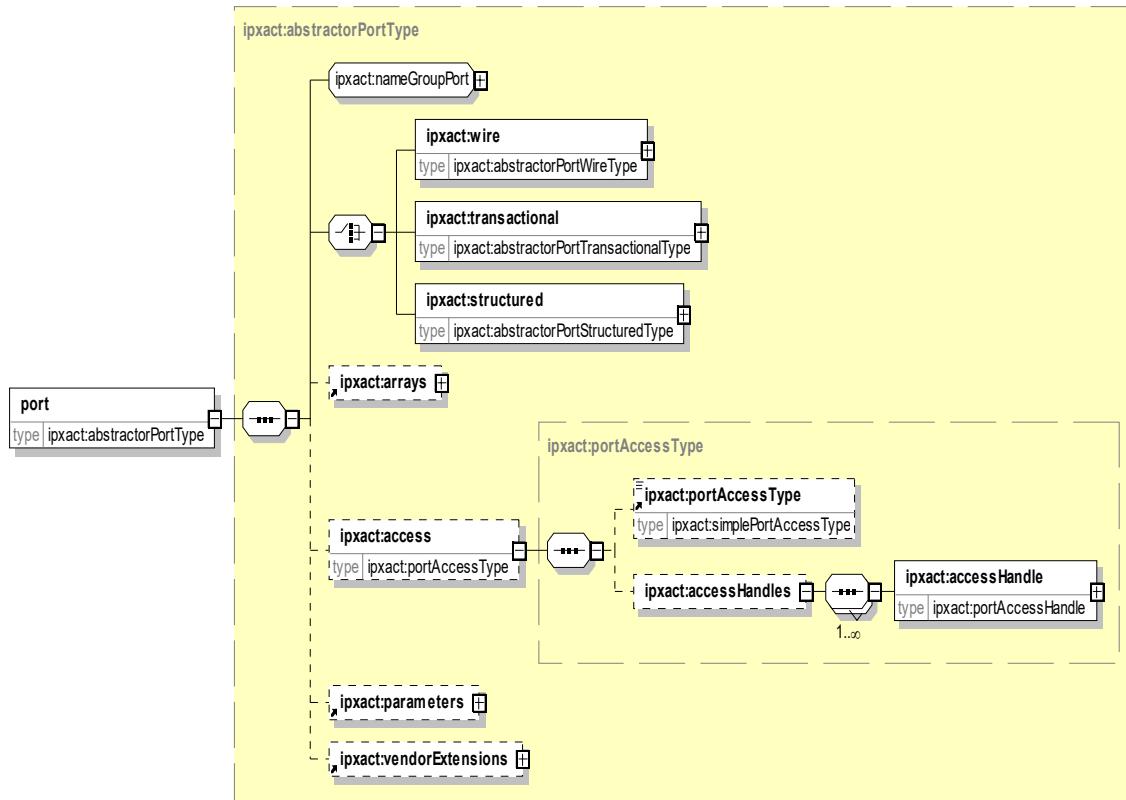
See also [SCR 6.26](#).

8.5 Abstracter ports

8.5.1 Schema

An abstractor's **ports** are almost identical to a component's **ports**. The access methods are the same for an abstractor or component port. The abstractor ports defined here differ from component ports only by the absence of **constraintSet** and/or **powerConstraints** elements because these elements are not needed for abstractors.

The following schema defines the information contained in the **ports** element, which may appear within an **abstractor**.



8.5.2 Description

The **ports** element defines an unbounded list of **port** elements. Each **port** element describe a single external port on the abstractor.

- nameGroupPort** group is defined in [C.19.5](#). The **name** elements shall be unique within the containing **ports** element.
- Each **port** shall be described as a **wire**, **transactional**, or **structured** port.
 - wire** defines ports that transport purely binary values or vectors of binary values. A wire port in an abstractor contains most of the same elements and attributes as a wire port in a component, except for the **constraintSet** element. See [8.6](#).
 - transactional** defines all other style ports, typically used for TLM. A transactional port in an abstractor contains all the same elements and attributes as a transactional port in a component. See [6.15.19](#).

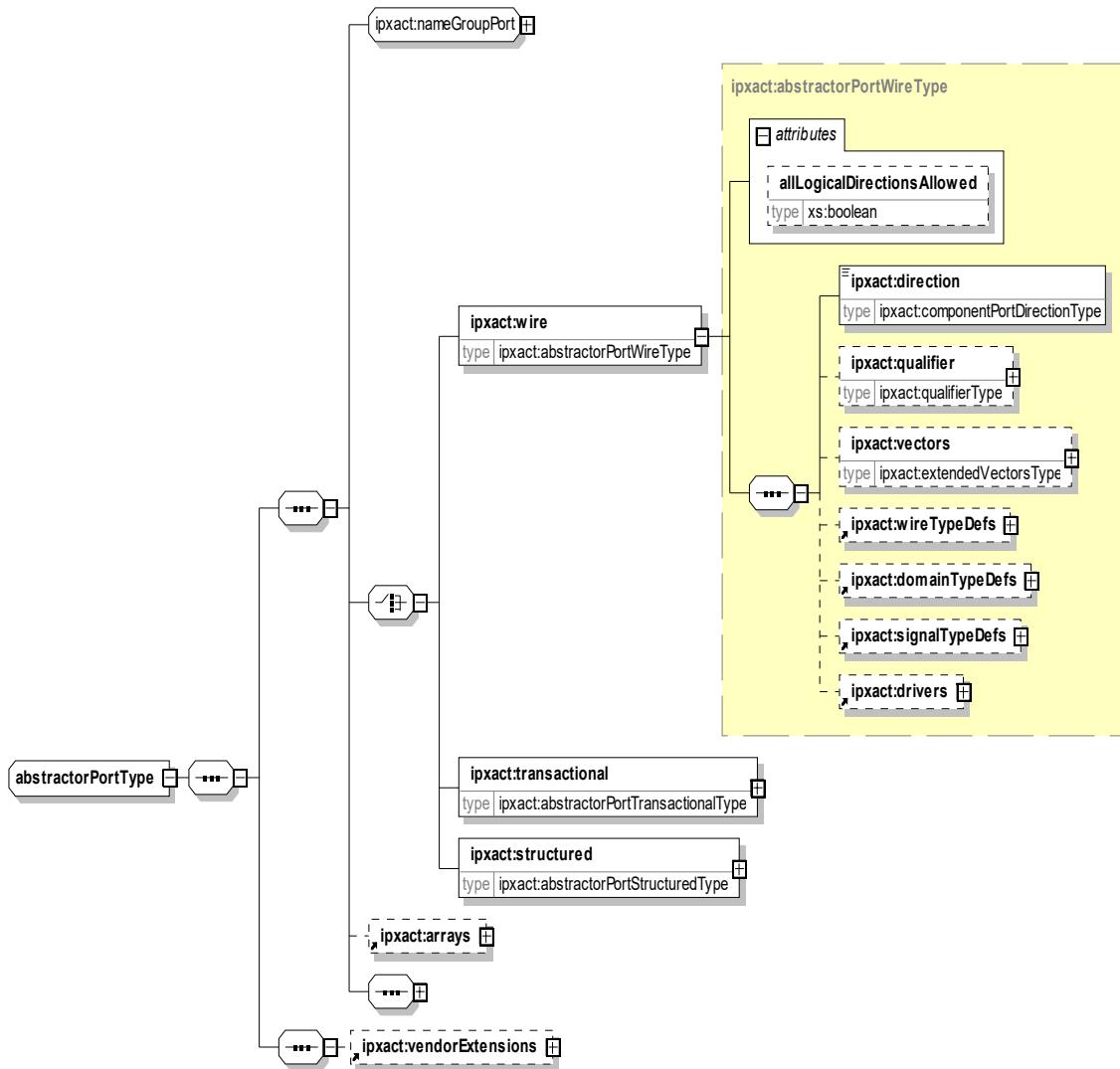
- 3) **structured** defines a port that is a structure, union, or SystemVerilog interface composed out of binary values.
- c) **arrays** (optional; type: configurableArrays, see [C.4](#)) specifies dimensions required to define an arrayed value type.
- d) **access** (optional) defines the access for a port.
 - 1) **portAccessType** (optional; default: **ref**) indicates to a netlister how to access the port. The **portAccessType** shall have one of two possible values **ref** or **ptr**. If **ref**, a netlister should access the port directly, and if **ptr**, it should access the port with a pointer.
 - 2) **accessHandles** (optional; type: **string**) indicates to a netlister the method to be used to access the object representing the port. This is typically a function call or array element reference in IEEE Std 1666 [\[B4\]](#) (SystemC). See [C.1.3](#).
- e) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See [C.27](#).

8.6 Abstractor wire ports

8.6.1 Schema

The abstractor **wire** ports defined here differ from component **wire** ports only by the absence of the **constraintSet** element and the **powerConstraints** element because implementation constraints are not needed for abstractors.

The following schema element defines the information contained in the **wire** element, which appears within an abstractor **port**.



8.6.2 Description

The **wire** element describes the properties for ports that are of a wire style. A port can come in two different styles, wire or transactional. A wire port can have four different signal representations, continuous-conservative, continuous-non-conservative, discrete, and digital. A digital wire port applies for all scalar types (e.g., VHDL `std_logic` and Verilog `wire`) and vectors of scalars. A digital wire port transports purely binary values or vectors of binary values.

- Scalar types in VHDL also include integer and enumeration values. Scalars in IP-XACT include only binary values that relate to a single wire in a hardware implementation.
- Since digital wire ports allow only binary values, they do not support tri-state or multiple strength values.

The **wire** element contains the following elements:

- **allLogicalDirectionsAllowed** (optional; type: `boolean`; default: `false`) attribute defines whether the port may be mapped to a port in an **abstractionDefinition** with a different direction. See [5.3](#).

- b) **direction** (mandatory) specifies the direction of this port: **in** for input ports, **out** for output ports, and **inout** for bidirectional and tri-state ports. **phantom** can also be used to define a port that exists only on the IP-XACT component, but not on the implementation referenced from the view.
- c) **qualifier** (optional) indicates which type of information this wire port carries; see [5.6](#).
- d) **vectors** (optional) specifies the dimensions for a non-scalar port; see [C.26.2](#).
- e) **domainTypeDefs** (optional) describes the ports domain as defined by the implementation; see [6.15.9](#).
- f) **signalTypeDefs** (optional) describes the ports signal representation as defined by the implementation; see [6.15.10](#).
- g) **wireTypeDefs** (optional) describes the ports type as defined by the implementation; see [6.15.11](#).
- h) **drivers** (optional) defines an unbounded list of driver elements, defining drivers that may be attached to this port if no other object is connected to this port. This allows the IP to define the default state of unconnected inputs. A wire style port may define a **driver** element for a port only if the direction of the port is **in** or **inout**. See also [6.15.12](#).

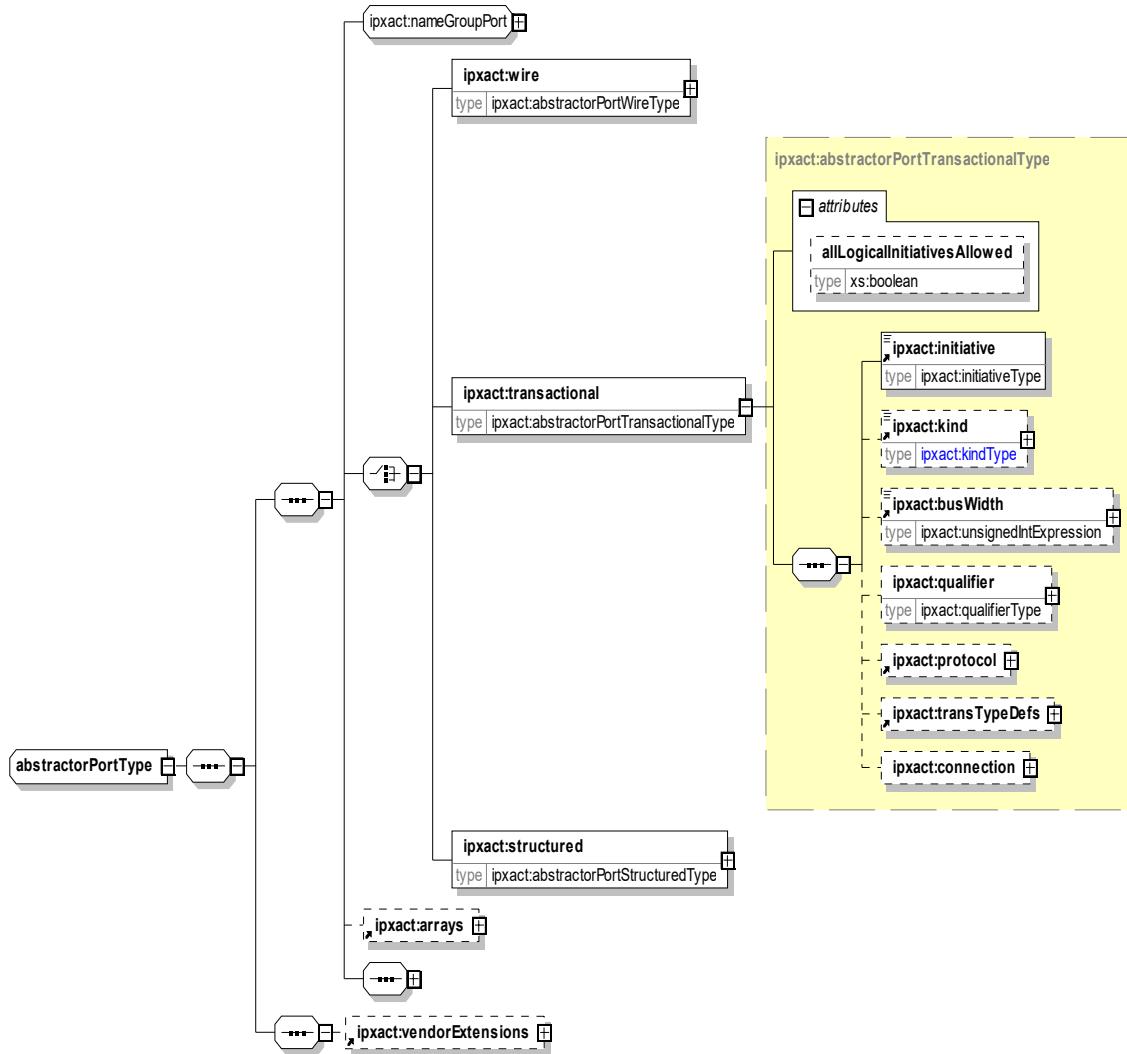
If multiple drivers are specified, the **range** element shall be used to indicate which bits are driven by which driver, and no overlap bit ranges may be specified.

See also [SCR 6.5](#), [SCR 6.6](#), [SCR 6.7](#), and [SCR 6.12](#).

8.7 Abstractor transactional port

8.7.1 Schema

The abstractor **transactional** ports defined here differ from component **transactional** ports in that the abstractor transactional port does not contain **powerConstraints** element. The following schema element defines the information contained in the **transactional** element, which appears within an abstractor **port**.



8.7.2 Description

The abstractor **transactional** element describes all other style of ports beyond wire and structured. **Transactional** ports are typically used for TLM. See also [6.15.19](#).

The abstractor **transactional** element contains the following elements:

- initiative** (mandatory) defines the type of access: **requires**, **provides**, **both**, or **phantom**.
 - For example, a SystemC `sc_port` should be defined with the **requires** initiative, since it requires a SystemC interface. A SystemC `sc_export` should be defined with the **provides** initiative, since it provides a SystemC interface.
 - both** indicates the type of access is both **requires** and **provides**.
 - phantom** indicates a phantom port is being defined. See [6.15.26](#).
- kind** (optional) specifies the transactional port's type. It can take one of the following `enum` values:

tlm_port, **tlm_socket**, **simple_socket**, **multi_socket**, or **custom**. When **custom**, a *name* can be specified in the **custom** (type: *string*) attribute. Also, the **tlm_port** should be used only for TLM1 notation; the other *enum* values should be used for TLM2 sockets.

- c) **busWidth** (optional; type: *unsignedIntExpression*, see [C.7.11](#)) specifies the data width in bits, e.g., 32 or 64.
- d) **qualifier** (optional) indicates which type of information this transactional port carries. See [5.6](#).
- e) **protocol** (optional) characterizes how the information is transported by the transactional port. See [6.15.20](#).
- f) **transTypeDefs** (optional) defines the port type expressed in the default language for this port. See [6.15.21](#).
- g) **connection** (optional) defines the number of legal connections for a port.
 - 1) **maxConnections** (optional; type: *unsignedIntExpression*, see [C.7.11](#); default: 0) indicating the maximum number of connections that this port supports. 0 indicates an unbounded number of legal connections.
 - 2) **minConnections** (optional; type: *unsignedIntExpression*, see [C.7.11](#); default: 1) indicating the minimum number of connections that this port supports.

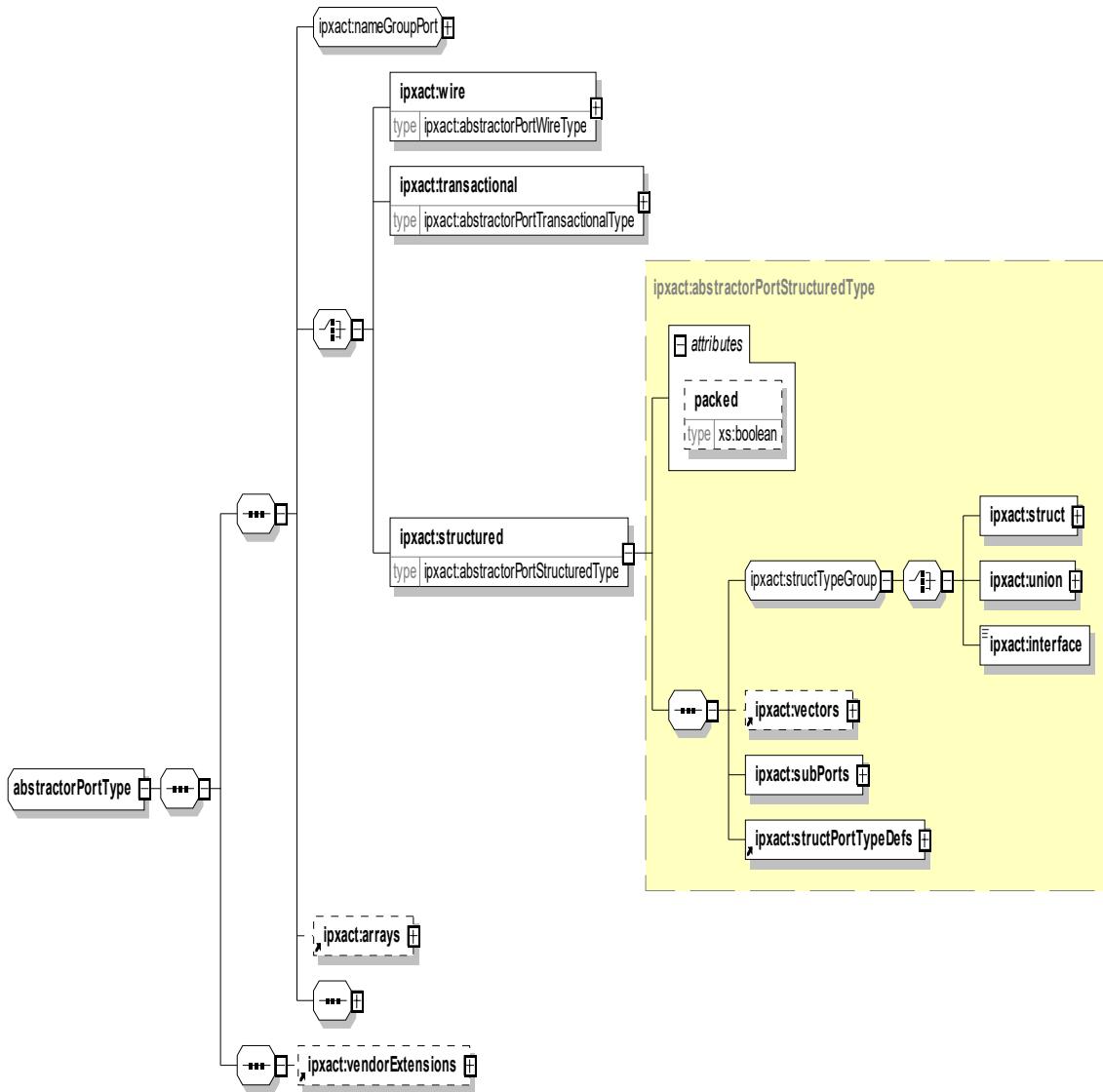
See also [SCR 6.2](#), [SCR 6.3](#), [SCR 6.4](#), [SCR 6.13](#), [SCR 6.21](#), and [SCR 6.22](#).

8.8 Abstractor structured ports

8.8.1 Schema

The abstractor **structured** ports defined here differ from component **structured** ports in that the abstractor **structured** port has abstractor wire ports as leafs rather than component wire ports.

The following schema element defines the information contained in the **structured** element, which appears within an abstractor **port**.



8.8.2 Description

The abstractor **structured** port defines a port that is a structure, union, or SystemVerilog interface composed out of binary values. See [6.15.23](#).

The abstractor **structured** element contains the following elements:

- structTypeGroup** (mandatory) is choice between elements **struct**, **union**, and **interface**. Struct indicates that the structured port is a structure of subports. Union indicates that the structured port is union of subports. Interface indicates that structured port is an (SystemVerilog) interface. Elements **struct** and **union** have an attribute **direction**. The value of direction shall match with the direction of the wire sub-ports of the structured port.
- vectors** (optional) define the dimensions of a **vector** when the parameter represents a packed array or bit-vector (where the **type** is **bit**). The **vector** element contains left and right elements. Vectors are also in component wire ports and abstractor ports. See [C.26.2](#).

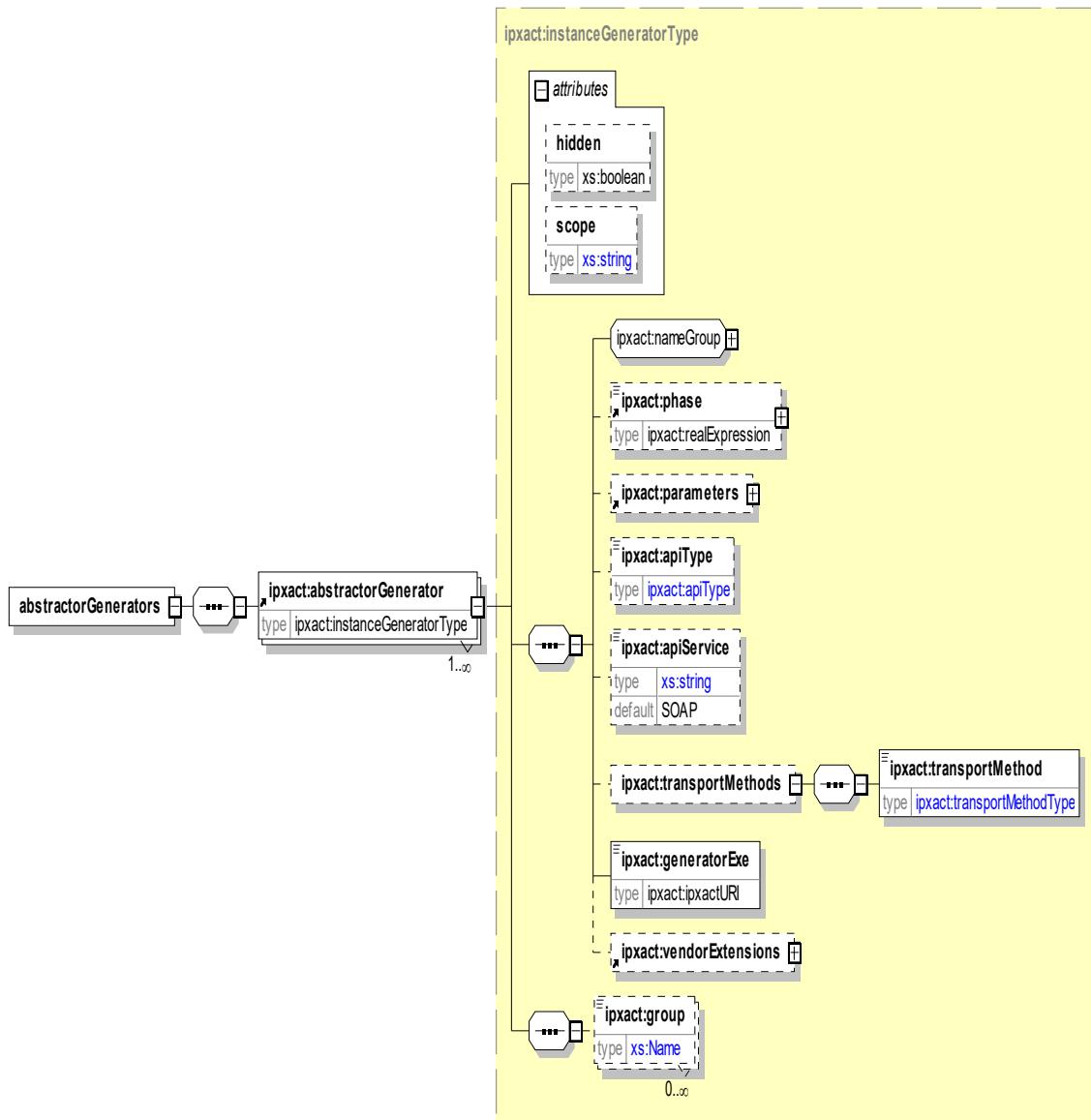
- c) **subPorts** (mandatory) contains a documentNameGroup, a choice between wire and structured, arrays, access, and vendorExtensions. See [6.15.24](#).
- d) **structPortTypeDefs** (mandatory) element contains one or more **structPortTypeDef** elements. See [6.15.25](#).

See [SCR 6.28](#).

8.9 Abstracter generators

8.9.1 Schema

The following schema defines the information contained in the **abstracterGenerators** element, which may appear within an **abstracter** object.



8.9.2 Description

The **abstractorGenerators** element contains an unbounded list of **abstractorGenerator** elements. Each **abstractorGenerator** element defines a generator that is assigned and may be run on this abstractor. The **abstractorGenerator** has exactly the same schema definition as a **componentGenerator**. See [6.16](#).

9. Type definitions descriptions

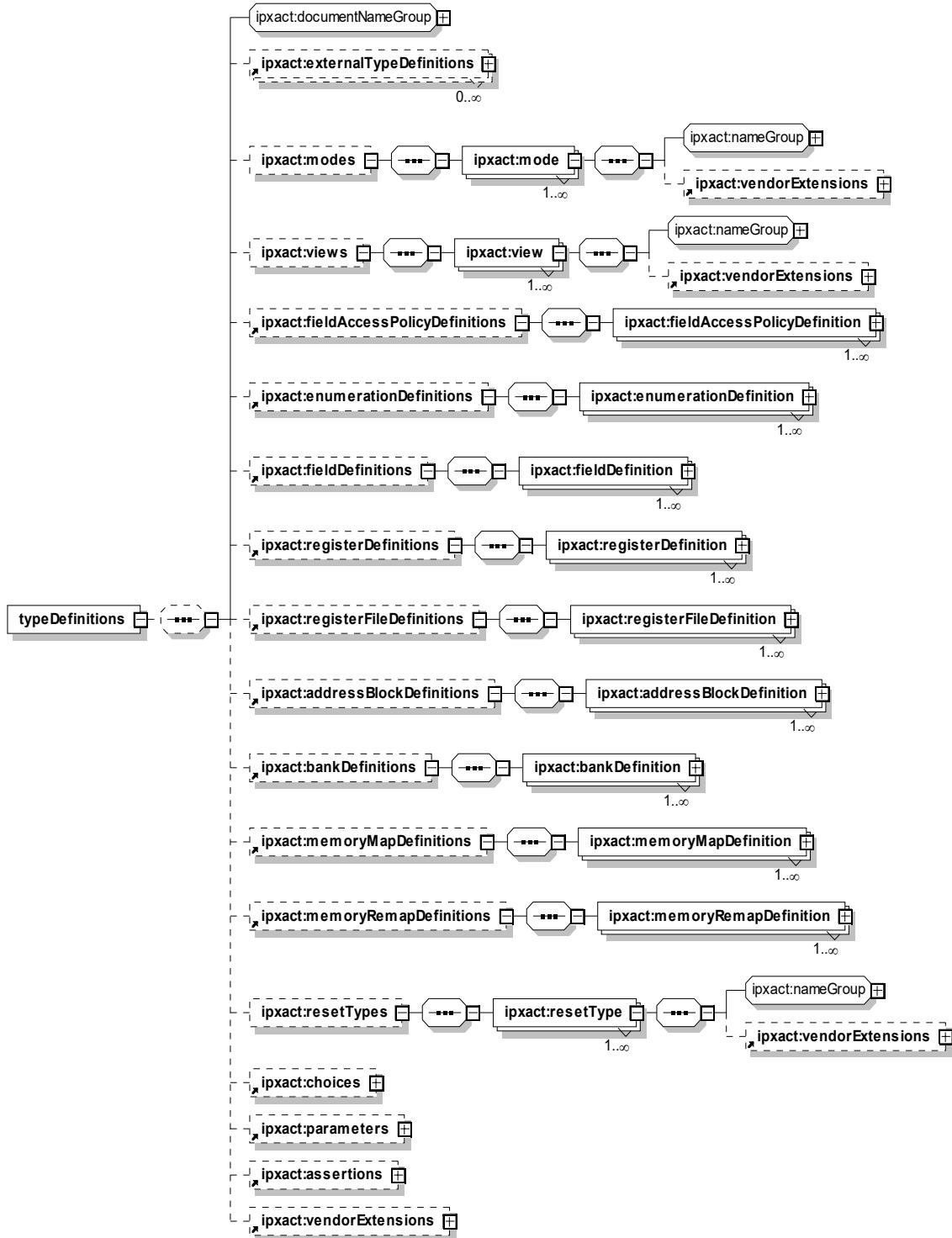
9.1 Type definitions

An IP-XACT *typeDefinitions* is the central placeholder for the definition of memory-related objects metadata. A typeDefinitions lists definitions of field access policies, enumerations, fields, registers, register files, address blocks, banks, memory maps, and memory remaps. These definitions can be configured and referenced in components and other typeDefinitions.

A **typeDefinitions** element may also contain **modes**, **views**, and **resetTypes**. These elements describe symbolic names that shall be linked to actual **modes**, **views**, and **resetTypes** in component **externalTypeDefinitions** elements for **typeDefinition** instances. See [6.2.2](#).

9.1.1 Schema

The following schema details the information contained in the **typeDefinitions** element, which is one of the top-level elements in the IP-XACT schema.



9.1.2 Description

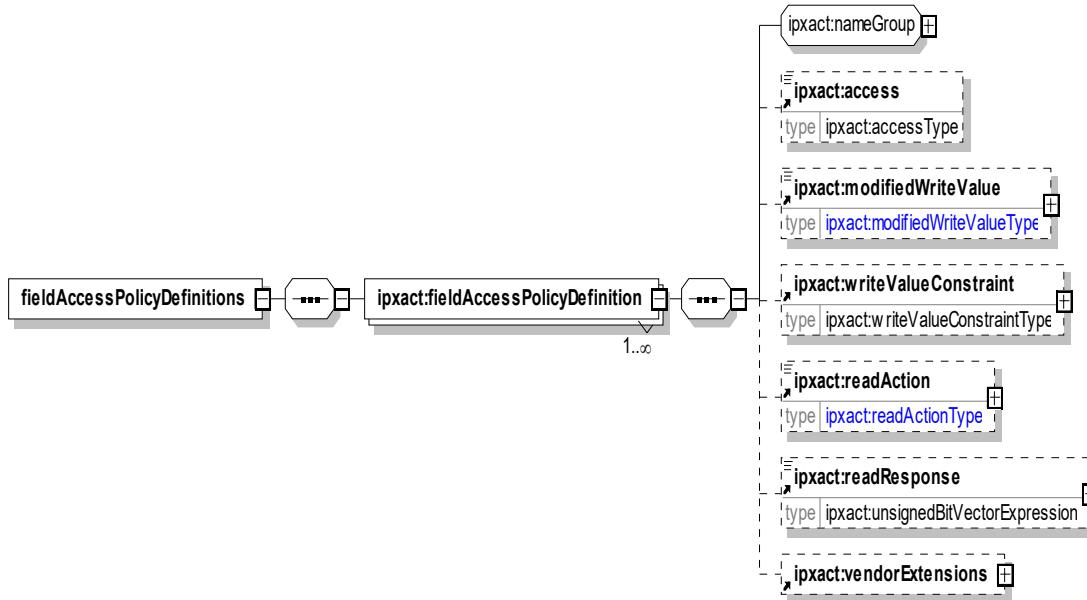
Each element of a **typeDefinitions** is detailed in the rest of this subclause; the main sections of a **typeDefinitions** are as follows:

- a) **documentNameGroup** describes **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. The **versionedIdentifier** or VLVN consists of four subelements for a top-level IP-XACT element. See [C.11](#).
- b) **externalTypeDefinitions** (optional) describes binding of external type definitions in these type definitions. See [6.2.2](#).
- c) **modes** (optional) describes operating modes that are referenced in these type definitions. Each mode contains a **nameGroup** and **vendorExtensions**.
- d) **views** (optional) describes views that are referenced in these type definitions. Each view contains a **nameGroup** and **vendorExtensions**.
- e) **fieldAccessPolicyDefinitions** (optional) describes field access policy definitions.
- f) **enumerationDefinitions** (optional) describes enumeration definitions.
- g) **fieldDefinitions** (optional) describes field definitions.
- h) **registerDefinitions** (optional) describes register definitions.
- i) **registerFileDefinitions** (optional) describes register file definitions.
- j) **addressBlockDefinitions** (optional) describes address block definitions.
- k) **bankDefinitions** (optional) describes bank definitions.
- l) **memoryMapDefinitions** (optional) describes memory map definitions.
- m) **memoryRemapDefinitions** (optional) describes memory remap definitions.
- n) **resetTypes** (optional) specifies user-defined reset types referenced within field reset elements. See [6.21](#).
- o) **choices** (optional) specifies multiple enumerated lists. These lists are referenced by other sections of this type definitions description. See [C.8](#).
- p) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to these type definitions. See [C.21](#).
- q) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.5](#).
- r) **vendorExtensions** (optional) contains any extra vendor-specific data related to these type definitions. See [C.27](#).

9.2 Field access policy definition

9.2.1 Schema

The following schema details the information contained in the **fieldAccessPolicyDefinition** element, which may appear as a element inside the **typeDefinitions** element.



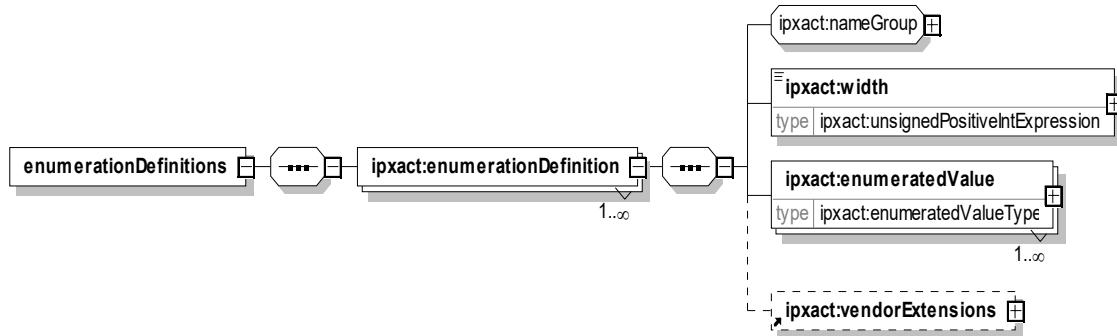
9.2.2 Description

- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- access** (optional) indicates the accessibility of the field. If this is not present, the **access** is inherited from the referencing register. See [6.14.9.2](#).
- modifiedWriteValue** (optional) element to describe the manipulation of data written to a field. See [6.14.9.2](#).
- writeValueConstraints** (optional) describes a set of constraint values that are the only values that can be written to this field. If **writeValueConstraint** is not present, no constraint values are defined for this field. See [6.14.10](#).
- readAction** (optional) describes an action that happens to a field after a read operation happens. See [6.14.9.2](#).
- readResponse** (optional) describes the value read after the read mux. See [6.14.9.2](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the field access policy definition. See [C.27](#).

9.3 Enumeration definition

9.3.1 Schema

The following schema details the information contained in the **enumerationDefinition** element, which may appear as a element inside the **typeDefinitions** element.



9.3.2 Description

- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- width** (mandatory; type: *unsignedPositiveIntExpression*, see [C.7.10](#)) describes the number of bits in the representation of this enumerated value.
- enumeratedValue** (mandatory) describes a field enumerated value using **name**, **width**, and **value**. See [6.14.12.2](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the enumeration definition. See [C.27](#).

9.4 Field definition

9.4.1 Schema

The following schema details the information contained in the **fieldDefinition** element, which may appear as a element inside the **typeDefinitions** element.



9.4.2 Description

- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- typeIdentifier** (optional; type: *Name*) indicates multiple fields elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in **fieldDefinitionGroup**.
- bitWidth** (mandatory; type: *unsignedPositiveIntExpression*, see [C.7.13](#)) is the width of the field, counting in bits.
- volatile** (optional; type: *boolean*; default: **false**) when **true** indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this field can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. If this element is not present, it is presumed to be **false**.
- resets** (optional) describes the reset values of the field. Each reset element defines the reset value for a given reset type. reset has the following subelements:
 - The **resetTypeRef** attribute (optional; type: *Name*) identifies the type of reset being defined. If specified, this should correspond to a user-defined reset in the **resetTypes** element (see [6.21](#)). If

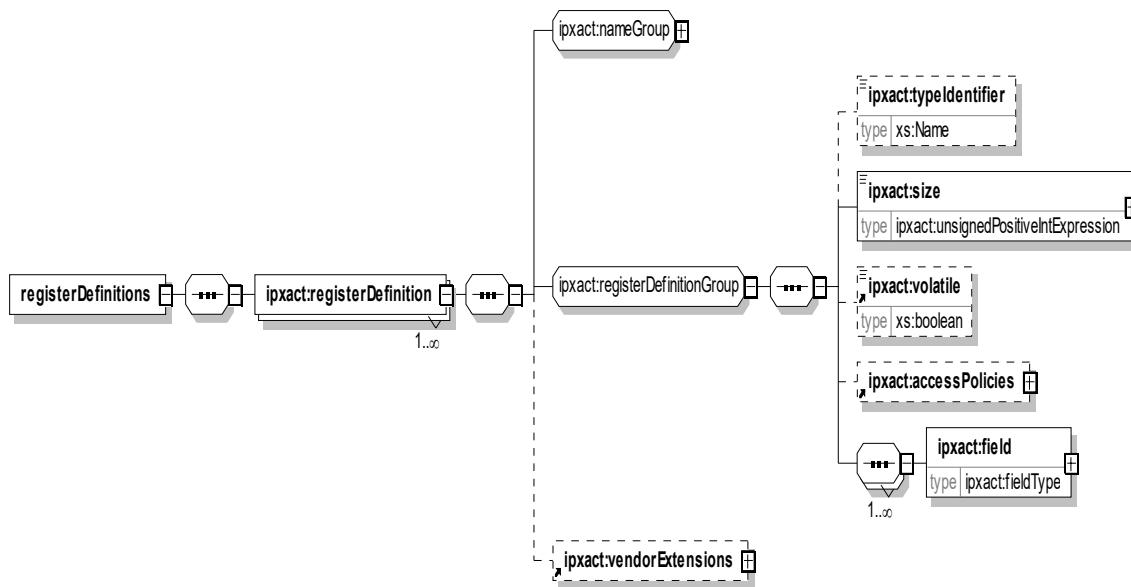
not specified, the default reset type is **HARD**. The resetTypeRef attribute value shall be unique within the containing field element.

- 2) **value** (mandatory; type: *unsignedBitVectorExpression*, see [C.7.10](#)) contains the actual reset value.
- 3) **mask** (optional; type: *unsignedBitVectorExpression*, see [C.7.10](#)) defines which bits of the register field have a known reset value. A 1 bit in the mask means the corresponding bit of the register field has a known reset value; a 0 bit means it does not. All bits of the value that correspond to 0 bits of the mask are ignored. The absence of a mask element is equivalent to a mask of the same size as the register field consisting of all 1 bits.
- f) **fieldData** group describes additional elements for a field. See [6.14.9](#).
- g) **vendorExtensions** (optional) contains any extra vendor-specific data related to the field definition. See [C.27](#).

9.5 Register definition

9.5.1 Schema

The following schema details the information contained in the **registerDefinition** element, which may appear as a element inside the **typeDefinition** element.



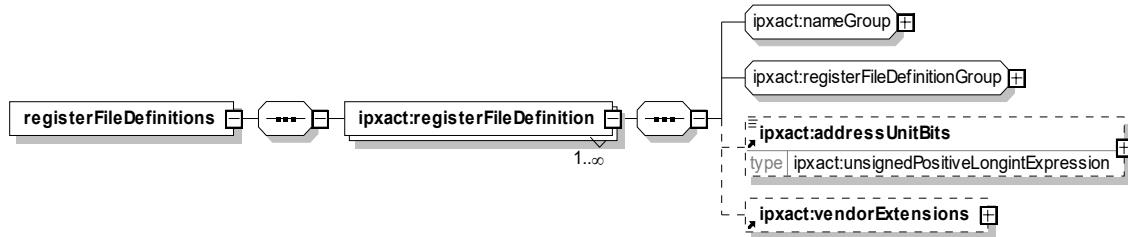
9.5.2 Description

- a) **nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- b) **registerDefinitionGroup** (mandatory) describes a register definition. See [6.14.3](#).
- c) **vendorExtensions** (optional) contains any extra vendor-specific data related to the registerDefinition. See [C.27](#).

9.6 Register file definition

9.6.1 Schema

The following schema details the information contained in the **registerFileDefinition** element, which may appear as a element inside the **typeDefinitions** element.



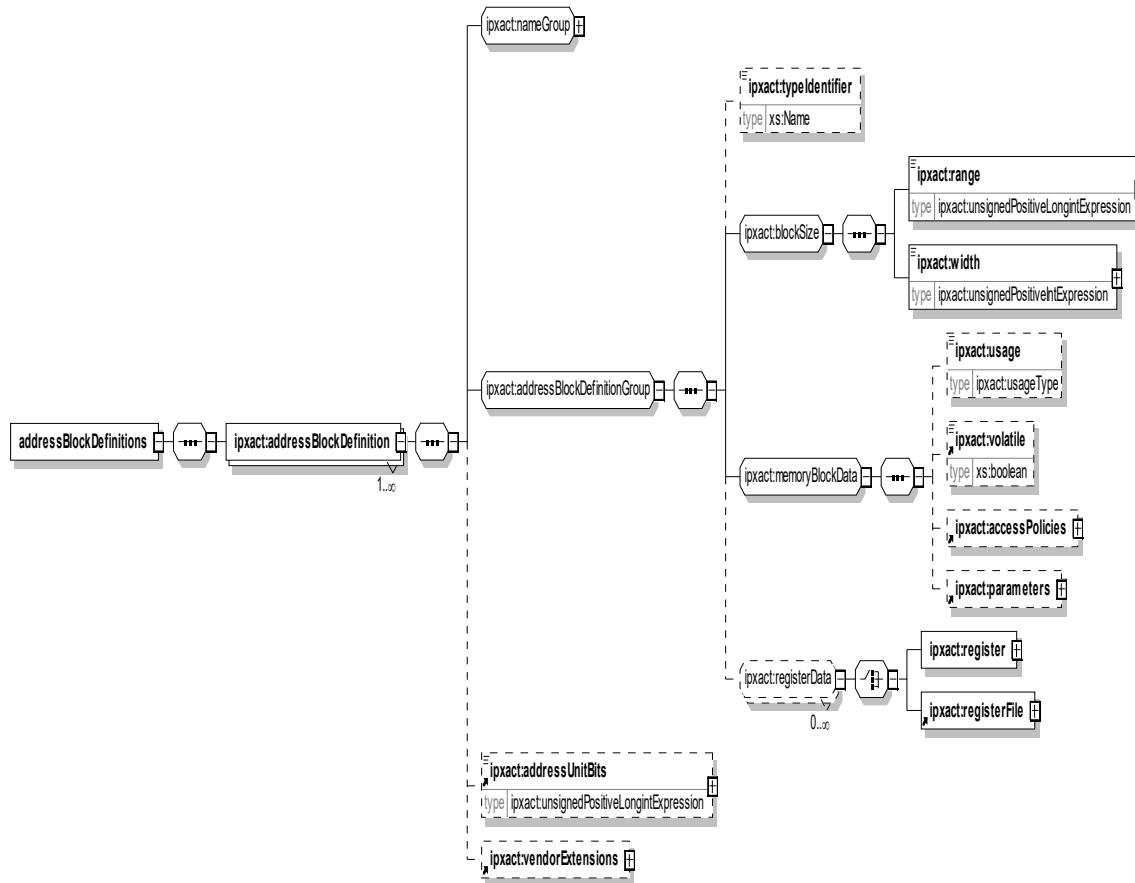
9.6.2 Description

- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- registerFileDefinitionGroup** (mandatory) describes a register file definition. See [6.14.7](#).
- addressUnitBits** element (optional; type: **unsignedPositiveLongintExpression**, see [C.7.14](#)) defines the number of data bits in each address increment of the memory map. This is required to allow the elements in the memory map to define items such as register offsets. If this element is not present, it is presumed to be **8**. See [6.12.1.2](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the register file definition. See [C.27](#).

9.7 Address block definition

9.7.1 Schema

The following schema details the information contained in the **addressBlockDefinition** element, which may appear as a element inside the **typeDefinitions** element.



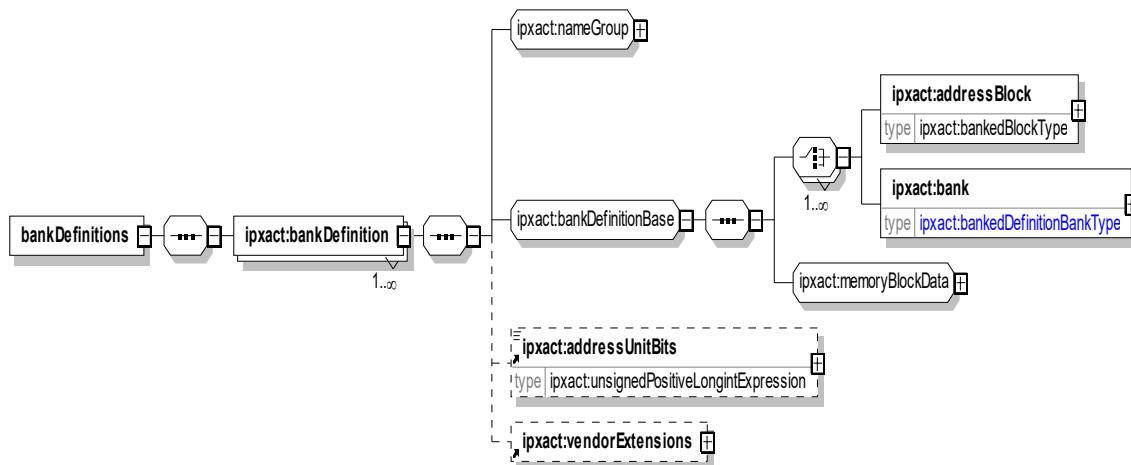
9.7.2 Description

- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- addressBlockDefinitionGroup** (mandatory) describes an address block definition. See [6.12.3](#).
- addressUnitBits** (optional; type: **unsignedPositiveLongintExpression**, see [C.7.14](#)) defines the number of data bits in each address increment of the address block. This is required to allow the elements in the address block to define items such as register offsets. If this element is not present, it is presumed to be 8. See [6.12.2](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the address block definition. See [C.27](#).

9.8 Bank definition

9.8.1 Schema

The following schema details the information contained in the **bankDefinition** element, which may appear as a element inside the **typeDefinitions** element.



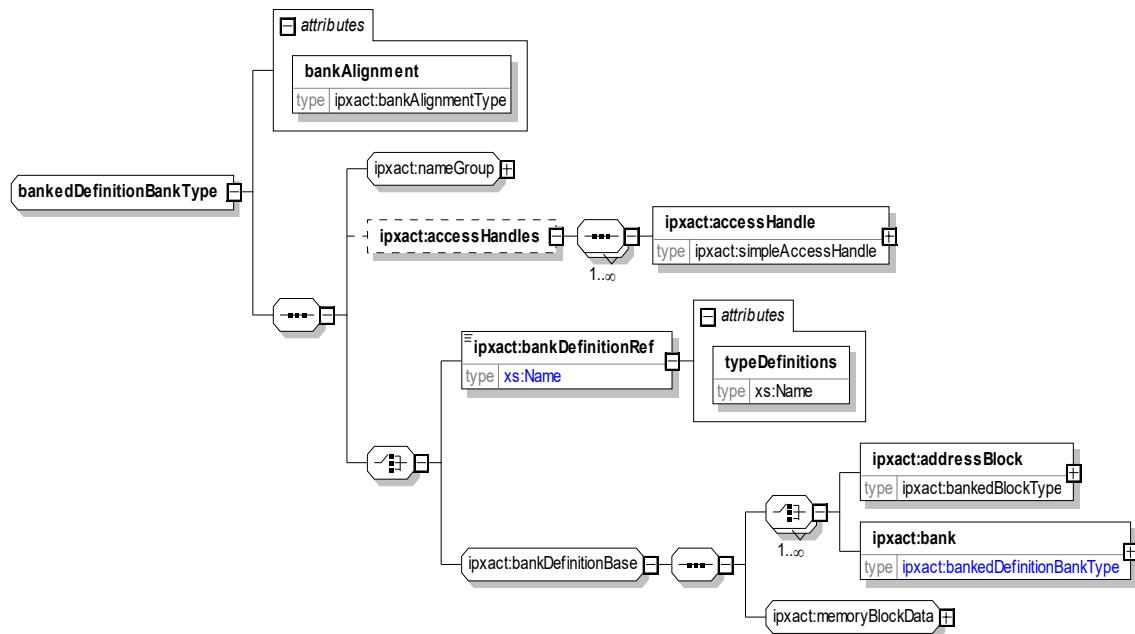
9.8.2 Description

- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- bankDefinitionBase** describes banked address blocks and/or banked banks and memory block data.
 - addressBlock** (mandatory; type: **bankedBlockType**) is an address block that makes up part of the bank. See [6.12.6](#).
 - bank** (mandatory; type: **bankedDefinitionBankType**) is a bank within the bank. This allows for complex configurations with nested banks. See [9.8.3](#).
 - memoryBlockData** (mandatory) is a collection of elements that contains further specification of **addressBlock** or **bank** elements See [6.12.4](#).
- addressUnitBits** (optional; type: **unsignedPositiveLongintExpression**, see [C.7.14](#)) defines the number of data bits in each address increment of the address block. This is required to allow the elements in the address block to define items such as register offsets. If this element is not present, it is presumed to be **8**. See [6.12.2](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the bank definition. See [C.27](#).

9.8.3 Banked definition bank type

9.8.3.1 Schema

The following schema details the information contained in the **bankedDefinitionBankType** element, which may appear as a element inside the **bankDefinitions** element.



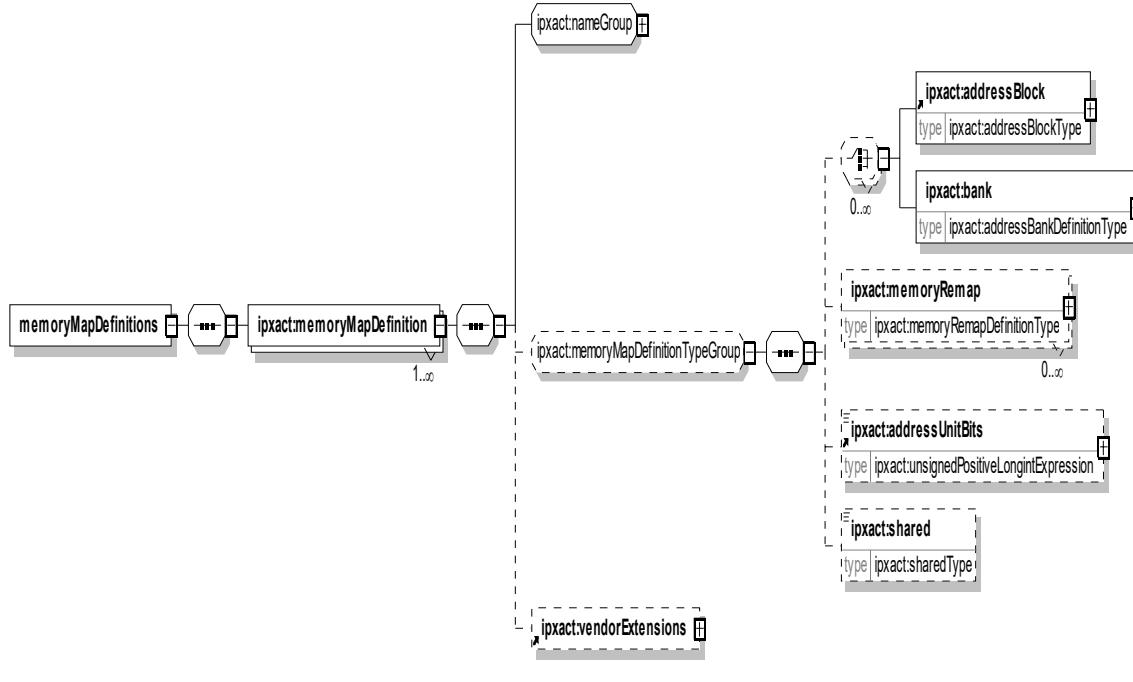
9.8.3.2 Description

- bankAlignment** (mandatory) attribute organizes the bank. See [6.12.5](#).
- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- accessHandles** (optional) specifies view-dependent naming for this bank within the corresponding view. See [C.1.1](#).
- A **bank** element inside a **bankDefinition** element contains either a **bankDefinitionRef** element or **bankDefinitionBase** group elements.
 - bankDefinitionRef** is reference to a bank name that is defined in a **typeDefinitions** that is bound in this **typeDefinitions** by an **externalTypeDefinition** that is referenced by the **typeDefinitions** attribute value. See [6.12.5](#).
 - bankDefinitionBase** describes banked address blocks and/or banked banks and memory block data.
 - addressBlock** (mandatory; type: **bankedBlockType**) is an address block that makes up part of the bank. See [6.12.6](#).
 - bank** (mandatory; type: **bankedDefinitionBankType**) is a bank within the bank. This allows for complex configurations with nested banks.
 - memoryBlockData** (mandatory) is a collection of elements that contains further specification of **addressBlock** or **bank** elements See [6.12.4](#).

9.9 Memory map definition

9.9.1 Schema

The following schema details the information contained in the **memoryMapDefinition** element, which may appear as a element inside the **typeDefinitions** element.



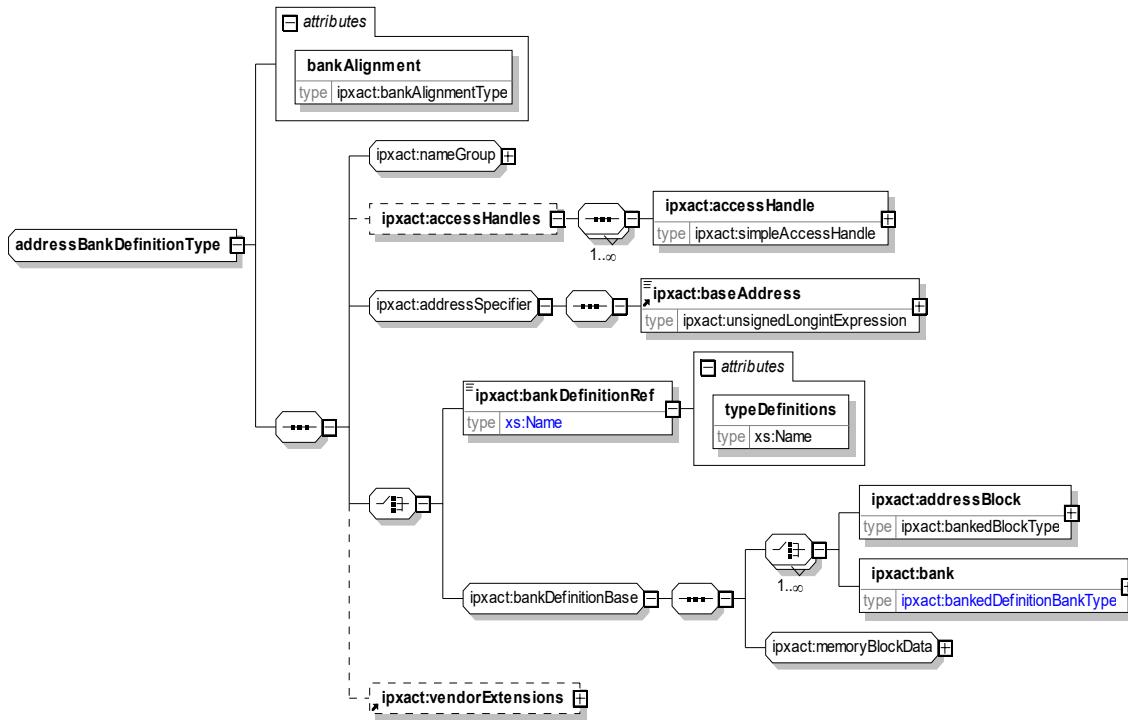
9.9.2 Description

- a) **nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- b) **memoryMapDefinitionTypeGroup** describes address blocks and/or banked banks and memory remaps.
 - 1) **addressBlock** (mandatory; type: **addressBlockType**) is a single address block. See [6.12.2](#).
 - 2) **bank** (mandatory; type: **addressBankDefinitionType**) represents a collection of address blocks and banks. See [9.8](#).
 - 3) **memoryRemap** (optional) describes additional address blocks and banks maps in a specific mode. See [6.13.1](#).
 - 4) **addressUnitBits** (optional; type: **unsignedPositiveLongintExpression**, see [C.7.14](#)) defines the number of data bits in each address increment of the memory map. This is required to allow the elements in the memory map to define items such as register offsets. If this element is not present, it is presumed to be 8. See [6.12.2](#).
 - 5) **shared** (optional; default: **undefined**) defines the sharing properties of the memory map. When the value is **yes**, the contents of the **memoryMap** are shared by all the references to this **memoryMap**. When the value is **no**, the contents of the **memoryMap** are not shared; and when the value is **undefined**, the sharing of the **memoryMap** is undefined.
- c) **vendorExtensions** (optional) contains any extra vendor-specific data related to the memory map definition. See [C.27](#).

9.9.3 Memory map definition bank

9.9.3.1 Schema

The following schema details the information contained in the **memoryMapDefinitionBank** element, which may appear as a element inside the **memoryMapDefinition** element.



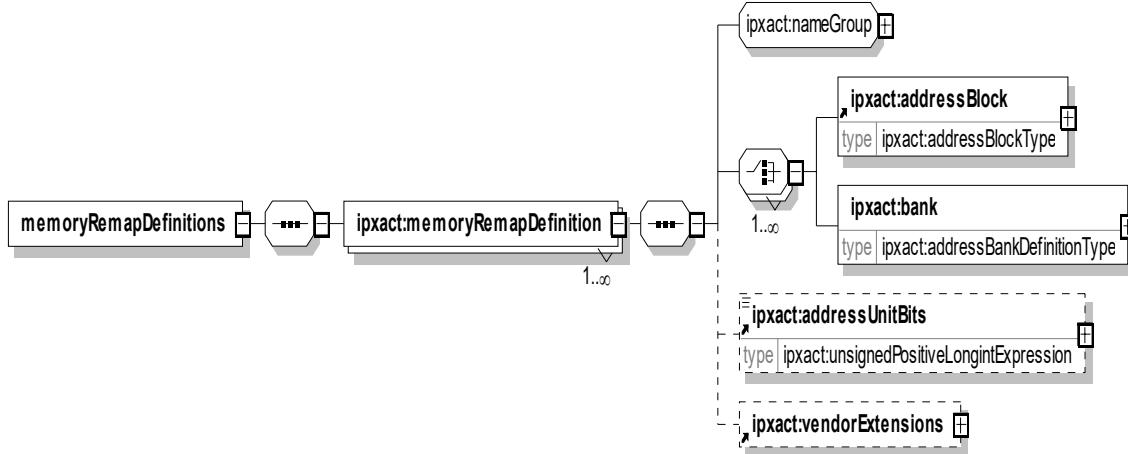
9.9.3.2 Description

- bankAlignment** (mandatory) attribute organizes the bank. See [6.12.5](#).
- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- accessHandles** (optional) specifies view-dependent naming for this bank within the corresponding view. See [C.1.1](#).
- addressSpecifier** group includes the following:
 - baseAddress** (mandatory; type: *unsignedLongintExpression*, see [C.7.12](#)) specifies the starting address of the block. The **baseAddress** is expressed in addressing units from the containing element's **addressUnitBits** element. See [6.12.5](#).
- A **bank** element inside a **memoryMapDefinition** element contains either a **bankDefinitionRef** element or **bankDefinitionBase** group elements.
 - bankDefinitionRef** is reference to a bank name that is defined in a **typeDefinitions** that is bound in this **typeDefinitions** by an **externalTypeDefinition** that is referenced by the **typeDefinitions** attribute value. See [6.12.5](#).
 - bankDefinitionBase** describes banked address blocks and/or banked banks and memory block data. See [9.8.3](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the bank definition. See [C.27](#).

9.10 Memory remap definition

9.10.1 Schema

The following schema details the information contained in the **memoryRemapDefinition** element, which may appear as a element inside the **typeDefinitions** element.



9.10.2 Description

- nameGroup** group defines any descriptive text for the containing element. See [C.19](#).
- addressBlock** (mandatory; type: *addressBlockType*) is a single address block. See [6.12.2](#).
- bank** (mandatory; type: *addressBankDefinitionType*) represents a collection of address blocks and banks. See [6.12.7](#).
- addressUnitBits** (optional; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) defines the number of data bits in each address increment of the memory map. This is required to allow the elements in the memory map to define items such as register offsets. If this element is not present, it is presumed to be 8. See [6.12.2](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the memory remap definition. See [C.27](#).

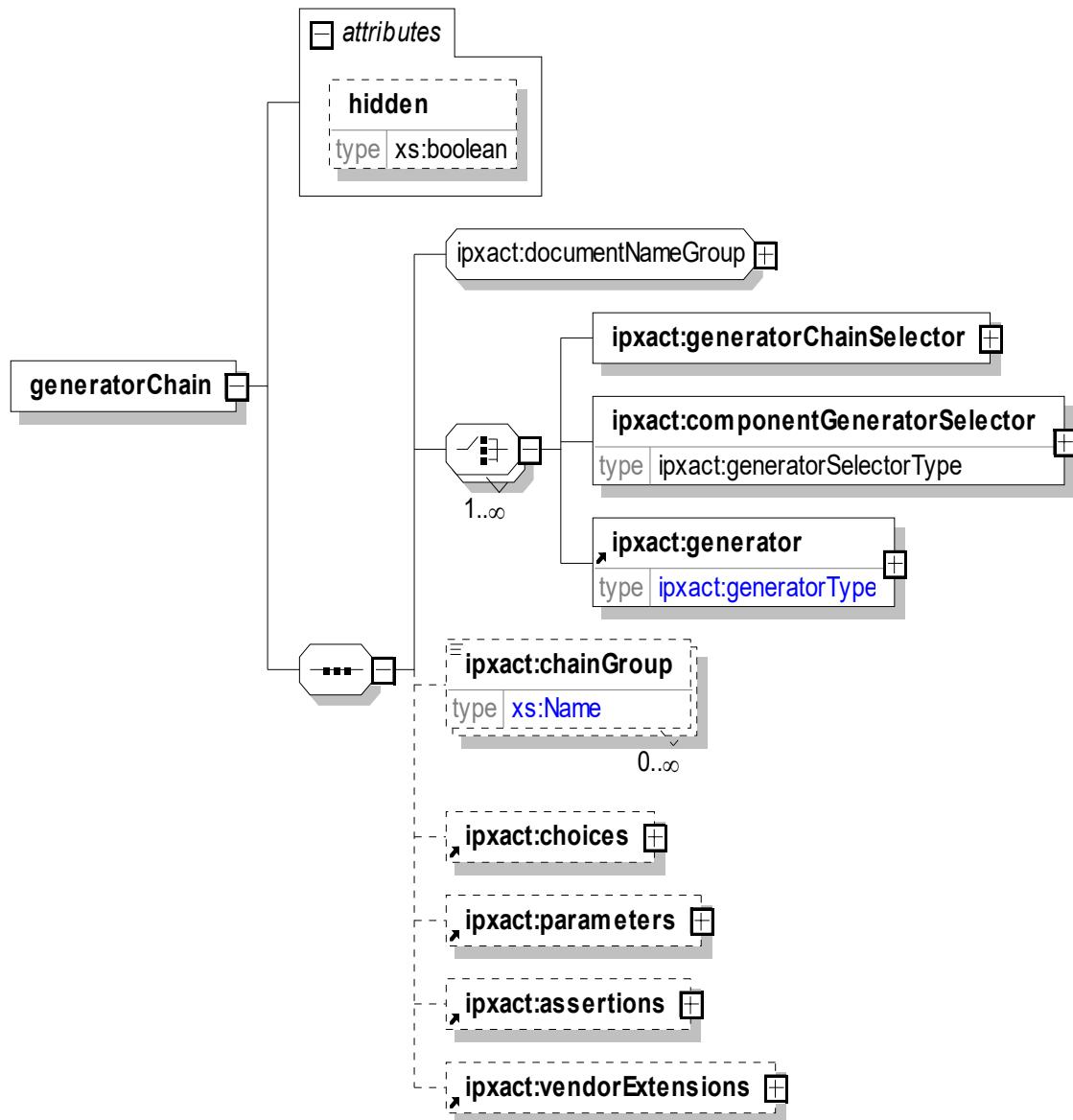
10. Generator chain descriptions

10.1 generatorChain

In IP-XACT, a design flow can be represented as a generator chain. A generator chain is an ordered sequence of named tasks. Each named task can be represented as a single generator or as another generator chain. This way, design flow hierarchies can be constructed and executed from within a given DE. The DE itself is responsible for understanding the semantics of the specified chain described in the generator chain description.

10.1.1 Schema

The following schema details the information contained in the **generatorChain** element, which is one of the top-level elements in the IP-XACT specification.



10.1.2 Description

The **generatorChain** element describes a single generator chain. The **generatorChain** element has a **hidden** attribute (optional; type: **boolean**; default: **false**) attribute that, when **true**, indicates this generator chain is not presented to the user of a DE. This may be the case if the chain is part of another chain and has no useful meaning when invoked as stand-alone. The **generatorChain** element contains the following elements:

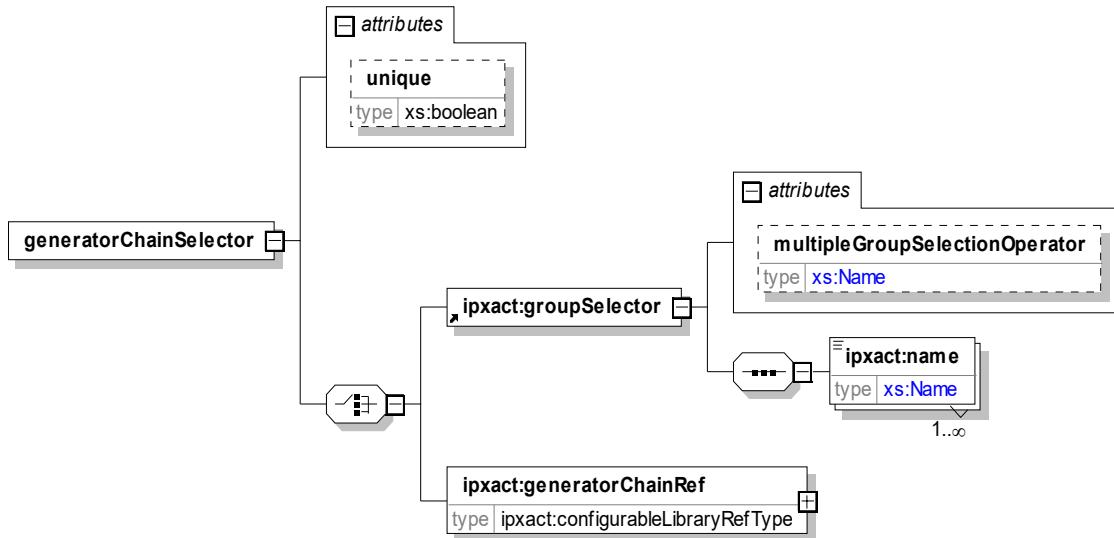
- a) **documentNameGroup** contains **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. See [C.11](#).
- b) A **generatorChain** also contains one or more of the following subelements:
 - 1) **generatorChainSelector** is a selection criteria for selecting one or more **generatorChains** or a reference to another **generatorChain** (see [10.2](#)).
 - 2) **componentGeneratorSelector** is a selection criteria for selecting one or more component generators (see [10.3](#)).
 - 3) **generator** defines the generator (see [10.4](#)).
- c) **chainGroup** (optional; type: *Name*) is an unbounded list of names to which this chain belongs. The group names are referenced in the **generatorChainSelector** element and can be used to organize the inclusion of generators.
- d) **choices** (optional) specifies multiple enumerated lists, which are referenced by other sections of this generator chain description. See [6.16](#).
- e) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this generator chain. See [C.21](#).
- f) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.5](#).
- g) **vendorExtensions** (optional) contains any extra vendor-specific data related to the **generatorChain**. See [C.27](#).

See also [SCR 1.10](#).

10.2 generatorChainSelector

10.2.1 Schema

The following schema defines the information contained in the **generatorChainSelector** element, which may appear within a **generatorChain**.



10.2.2 Description

The **generatorChainSelector** element defines which generator(s) to invoke based on a selection criteria. The **generatorChainSelector** element contains a **unique** (optional; type: **boolean**; default: **false**) attribute that, when **true**, indicates the generatorChainSelector shall resolve to a single generator. If more than one generator is selected, the DE shall resolve the selection to a single generator.

The **generatorChainSelector** element can specify the selection criteria in one of two ways: as a selection based on the **chainGroup** names via the **groupSelector** element or as a direct VLVN reference via the **generatorChainRef** element. The **generatorChainSelector** element shall contain one of the **groupSelector** or **generatorChainRef** elements.

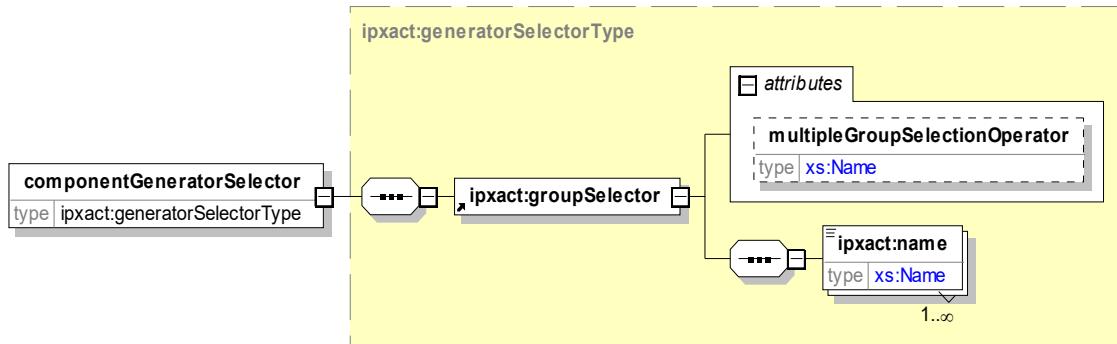
- a) **groupSelector** is a container for an unbounded list of chain group **name** elements.
 - 1) When more than one **name** element is specified, the **multipleGroupSelectorOperator** (optional; type: **Name**; default: **or**) attribute can specify if the selection applies when *one* of the generator group names matches (**multipleGroupSelectorOperator** equals **or**) or *all* the generator group names match (**multipleGroupSelectorOperator** equals **and**).
 - 2) **name** (mandatory; type **Name**) is an unbounded list of selection names. The names are compared to the **generatorChain/chainGroup** elements within all generator chains visible to the DE.
- b) **generatorChainRef** (type: **configurableLibraryRefType**, see [C.10](#)) specifies a reference to another generator chain description for inclusion in this generator chain.

See also [SCR 1.8](#).

10.3 generatorChain component selector

10.3.1 Schema

The following schema defines the information contained in the **componentGeneratorSelector** element, which may appear within a **generatorChain**.



10.3.2 Description

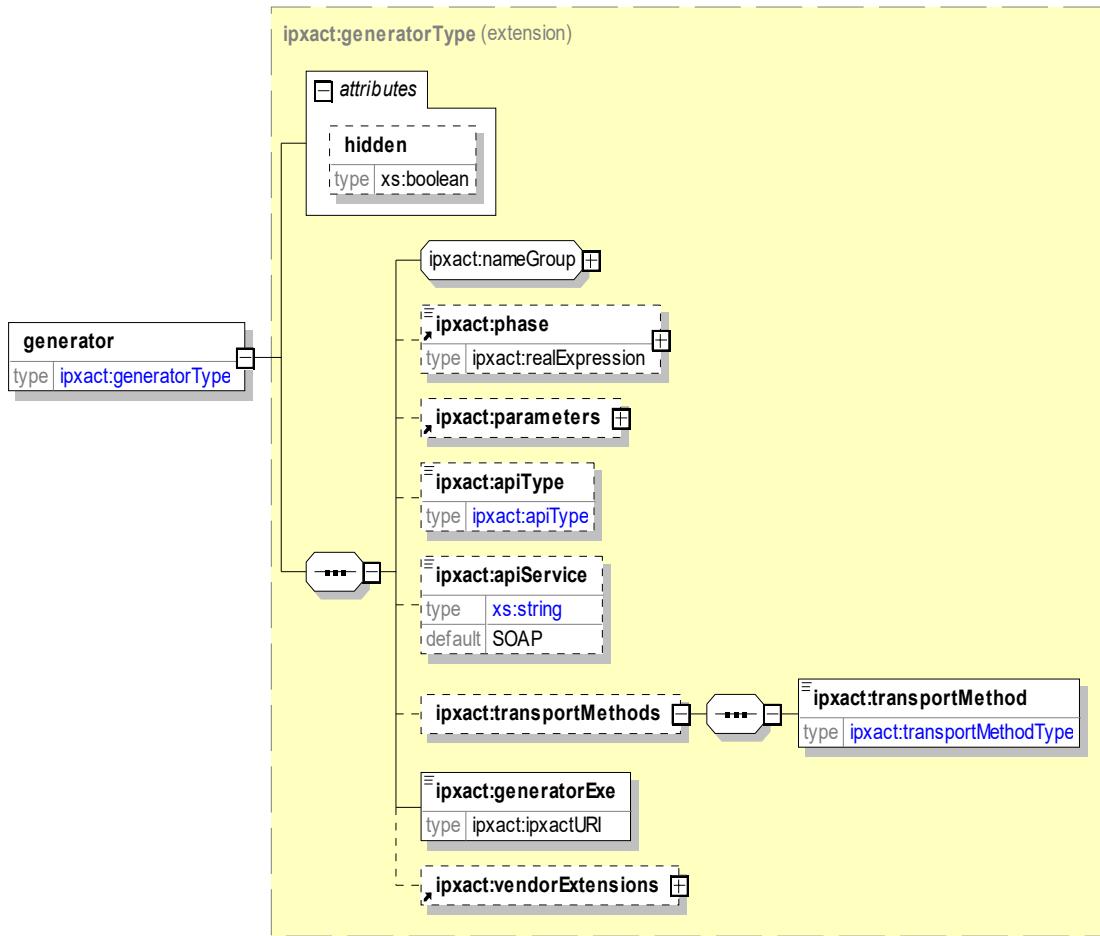
Similar to the **generatorChainSelector**, **componentGeneratorSelector** selects a component generator or a list of component generators based on the assigned group name. The **componentGeneratorSelector** contains the **groupSelector** element.

- a) **groupSelector** (mandatory) is a container for an unbounded list of chain group **name** elements.
 - 1) When more than one **name** element is specified, the **multipleGroupSelectorOperator** (optional; type: *Name*; default: **or**) attribute can specify if the selection applies when *one* of the generator group names matches (**multipleGroupSelectorOperator** equals **or**) or *all* the generator group names match (**multipleGroupSelectorOperator** equals **and**).
 - 2) **name** (mandatory; type: *Name*) is an unbounded list of selection names. The names are compared to the **componentGenerator/group** elements within all components in the current design.

10.4 generatorChain generator

10.4.1 Schema

The following schema defines the information contained in the **generator** element, which may appear within a **generatorChain**.



10.4.2 Description

The **generator** element defines a specific generator executable. The **generator** element contains has a **hidden** attribute (optional; type: *boolean*; default: *false*) attribute that, when *true*, indicates this generator shall not be run as a stand-alone generator and is required to be run as part of a chain. This generator is not presented to the user. If *false*, this generator may be run as a stand-alone generator or in a generator chain.

The **generator** contains the following elements:

- nameGroup** group is defined in [C.19](#).
- phase** (optional; type: *realExpression*, see [C.7.3](#)) determines the sequence in which generators are run. Generators are run in order starting with zero (0). If two generators have the same phase numbers, the order shall be interpreted as not important, and the generators can be run in any order. If no phase number is given, the generator is considered in the “last” phase, and these generators are run in any order after the last generator with a phase number. The **phase** element shall be a positive number.

- c) **parameters** (optional) specifies any **generator** type parameters. See [C.21](#).
- d) **apiType** (optional, type: **apiType**, default: **TGI_2022_BASE**) indicates the type of application programmers interface (API) used by the generator. Allowed values include the following:
 - 1) **TGI_2022_BASE** indicates a generator that communicates using the base TGI API defined for this standard.
 - 2) **TBGI_2022_EXTENDED** indicates a generator that communicates using the base and extended TGI APIs defined for this standard.
 - 3) **TGI_2014_BASE** indicates a generator that communicates using the base TGI API defined for IEEE Std 1685-2014. This API is not part of this standard and does not need to be supported by IP-XACT-enabled DES.
 - 4) **TGI_2014_EXTENDED** indicates a generator that communicates using the base and extended TGI APIs defined for IEEE Std 1685-2014. These APIs are not part of this standard and do not need to be supported by IP-XACT-enabled DES.
 - 5) **TGI_2009** indicates a generator that communicates using the TGI API defined for IEEE Std 1685-2009. This API is not part of this standard and does not need to be supported by IP-XACT-enabled DEs.
 - 6) **none** indicates the generator does not communicate with the DE using an API defined in this standard.
- e) **apiService** (optional) describes which transport service is used, i.e., SOAP or REST. Default is SOAP.
- f) **transportMethods** (optional) defines alternate transport protocols that this generator can support. The default transport protocol is HTTP if this element is not present.
- g) **transportMethod** specifies the alternate transport protocol. This element is an enumerated list of only one element file.
- h) **generatorExe** (mandatory; type *ipxactURI*) contains an absolute or relative (to the location of the containing description) path to the generator executable. The path may also contain environment variables from the host system, which are used to abstract the location of the generator.
- i) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **generator**. See [C.27](#).

11. Design configuration descriptions

11.1 Design configuration

An IP-XACT *design configuration* is a placeholder for additional configuration information of a design or generator chain description. Design configuration information is used to support configuration of hierarchical designs, for transferring designs between design environments and automating generator chain execution for a design, by storing information that would otherwise have to be re-entered by the designer.

The *design configuration description* contains the following configuration information:

- Configuration values for parameters defined in generators within generator chains; this information is not referenced via the design description.
- Active view or current view selection for instances in the design description along with configuration values for view parameters within the component instances.
- Configuration information for interconnections between the same bus types with differing abstraction types (i.e., abstractor reference, parameter configuration, and view selection). See also [Clause 8](#).

A design configuration applies to a single design, but a design may have multiple design configuration descriptions.

11.2 designConfiguration

11.2.1 Schema

The following schema details the information contained in the **designConfiguration** element, which is one of the top-level elements of the schema.



11.2.2 Description

The **designConfiguration** element details the configuration for a design or generator chain description. The **designConfiguration** element contains the following mandatory and optional elements:

- documentNameGroup** describes **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. The **versionedIdentifier** or VLVN consists of four subelements for a top-level IP-XACT element. See [C.28](#).
- designRef** (optional; type: **libraryRefType**, see [C.18](#)) specifies the design description for this design configuration.
 NOTE—It is recommended instead to pursue this functionality by using the **designInstantiation** element in a **component**.
- generatorChainConfiguration** (optional; type: **configurableLibraryRefType**, see [C.10](#)) documents a generator chain VLVN and an unbounded list of configuration values for parameters within the referenced generator chain.

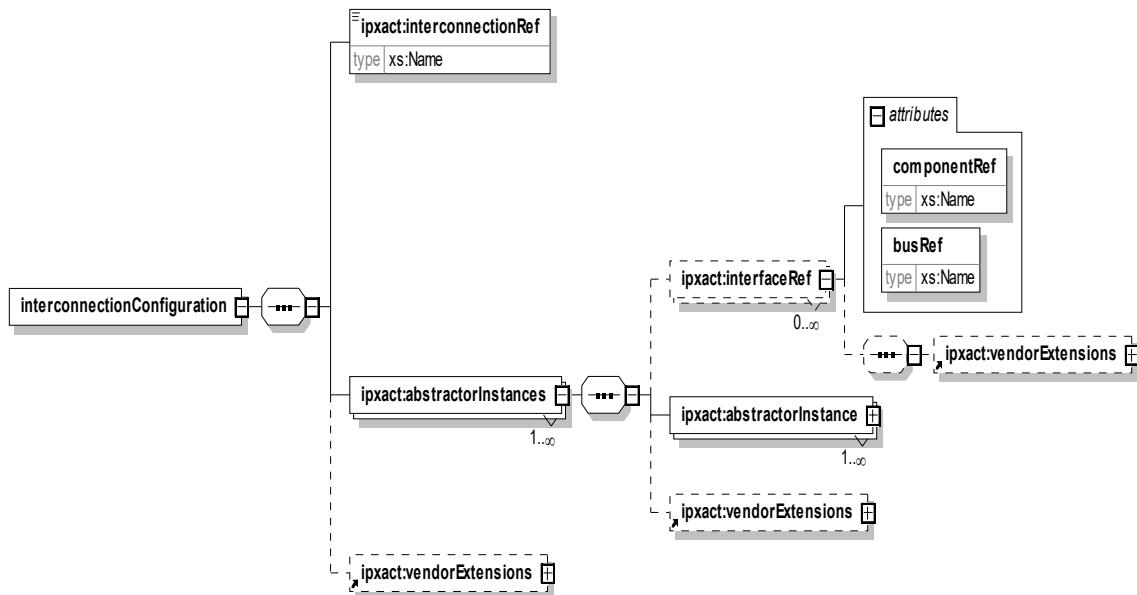
- d) **interconnectionConfiguration** (optional) is an unbounded list of information associated with interface interconnections. Any abstractors required for the connection of two interfaces are specified here. See [11.3](#).
- e) **viewConfiguration** (optional) defines the selected view and view configuration for an instance of the design. See [11.5](#).
- f) **choices** (optional) specifies multiple enumerated lists. These lists are referenced by other sections of this design configuration description. See [C.8](#).
- g) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this design configuration. See [C.21](#).
- h) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.5](#).
- i) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design configuration. See [C.27](#).

See also [SCR 1.5](#), [SCR 1.7](#), [SCR 1.10](#), [SCR 1.16](#), [SCR 1.17](#), [SCR 5.10](#), [SCR 13.1](#), [SCR 13.2](#), and [SCR 13.3](#).

11.3 interconnectionConfiguration

11.3.1 Schema

The following schema defines information contained in **interconnectionConfiguration** element, which may appear as an element inside the **designConfiguration** element.



11.3.2 Description

The **interconnectionConfiguration** element contains information about the **abstractors** used to resolve connections between interfaces having the same **busDefinition** types, but different **abstractionDefinition** types. The **interconnectionConfiguration** element contains the following elements:

- a) **interconnectionRef** (mandatory; type: *Name*) contains a reference to a design **interconnection/name** or a design **monitorInterconnection/name**. All **interconnectionRef** elements shall be unique within the containing design configuration description.

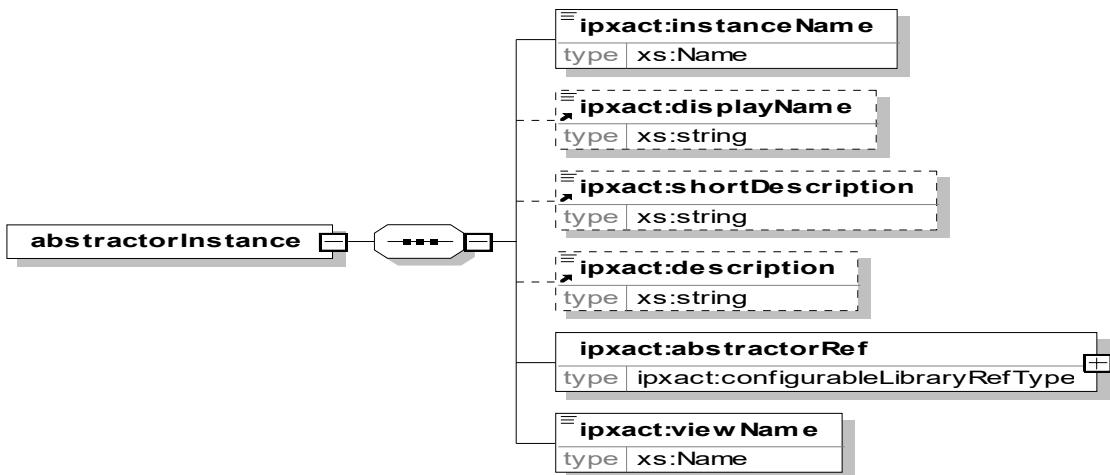
- b) **abstractorInstances** (mandatory) contains an unbounded list of **abstractor** elements. Multiple abstractors elements are allowed to enable insertion of different abstractor chains within a broadcast interface. If there are multiple abstractors elements, each is required to reference a unique interface via the **componentRef** and **interfaceRef** attributes in the **interfaceRef** element. If an **abstractors** element contains no **interfaceRef** element, it is presumed the contained chain of abstractors applies to all interfaces not explicitly referenced in other abstractors elements. The **abstractors** element contains the following mandatory and optional elements:
- 1) **interfaceRef** (optional) defines the broadcast endpoint for this abstractor chain. It contains the following subelements:
 - i) **componentRef** (mandatory; type: *Name*) refers to a component instance name.
 - ii) **busRef** (mandatory; type: *Name*) refers to a component bus interface name.
 - 3) **abstractorInstance** (mandatory) is an ordered list for chaining the abstractors together to bridge from one abstraction to another. See [11.4](#).
 - 4) **vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See [C.27](#).
- c) **vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See [C.27](#).

See also [SCR 1.13](#), [SCR 3.7](#), [SCR 3.11](#), [SCR 3.12](#), [SCR 3.13](#), [SCR 3.14](#), [SCR 3.18](#), [SCR 5.11](#), and [SCR 13.4](#).

11.4 abstractorInstance

11.4.1 Schema

The following schema defines information contained in **abstractorInstance** element, which may appear as an element inside the **interconnectonConfiguration** element.



11.4.2 Description

The **abstractorInstance** element is an ordered list for chaining the abstractors together to bridge from one abstraction to another. This element has the following subelements:

- a) **instanceName** (mandatory; type: *Name*) assigns a unique name for this instance of the abstractor in this design. The value of this element shall be unique inside the **designConfiguration** and the referenced **design** element.

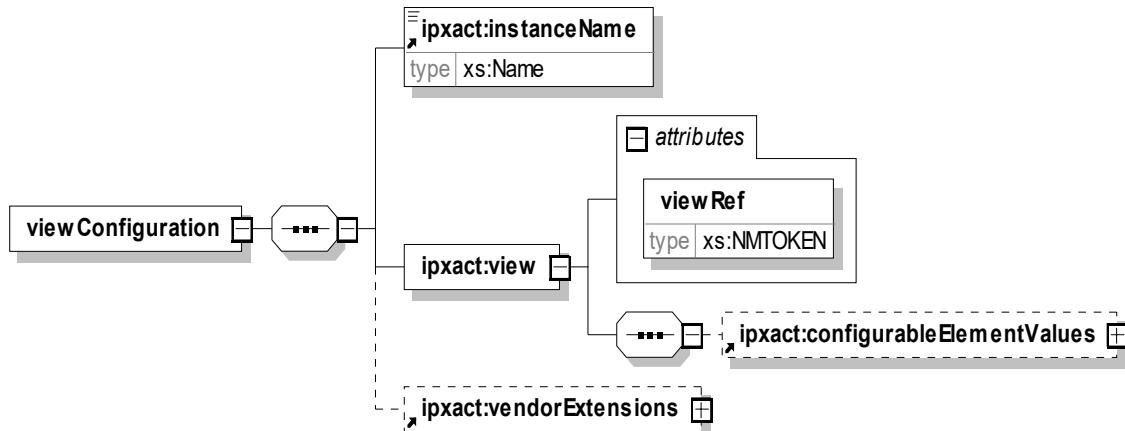
- b) **displayName** (optional; type: *string*) allows a short descriptive text to be associated with the containing element. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- c) **shortDescription** is a more compact textual description of the containing element compared to ip-xact:description.
- d) **description** (optional; type: *string*) allows a textual description of the instance.
- e) **abstractorRef** (mandatory; type: *configurableLibraryRefType*, see [C.10](#)) is a reference to an abstractor description for this abstractor instance.
- f) **viewName** (mandatory; type: *Name*) defines the selected view for the abstractor instance.

See also [SCR 1.13](#), [SCR 3.7](#), [SCR 3.11](#), [SCR 3.12](#), [SCR 3.13](#), [SCR 3.14](#), [SCR 5.11](#), and [SCR 13.4](#).

11.5 viewConfiguration

11.5.1 Schema

The following schema defines information contained in **viewConfiguration** element, which may appear as an element inside the **designConfiguration** element.



11.5.2 Description

The **viewConfiguration** element defines the active view and view configuration values for an instance of the design. The **viewConfiguration** element contains the following elements:

- a) **instanceName** (mandatory; type: *Name*) specifies the component instance name for which the view is being selected. This instance name shall be unique within the containing design configuration description.
- b) **view** (mandatory) defines the current valid view for the selected component instance.
 - 1) **viewRef** (mandatory; type: *NMTOKEN*) specifies the component view. See [6.15.1.2](#).
 - 2) **configurableElementValues** (optional) specifies the configuration values associated with parameters within the **componentInstantiation** and **designConfigurationInstantiation** referenced by the component view specified by the **viewRef**. See [C.9](#).
- c) **vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See [C.27](#).

See also [SCR 5.9](#).

12. Catalog descriptions

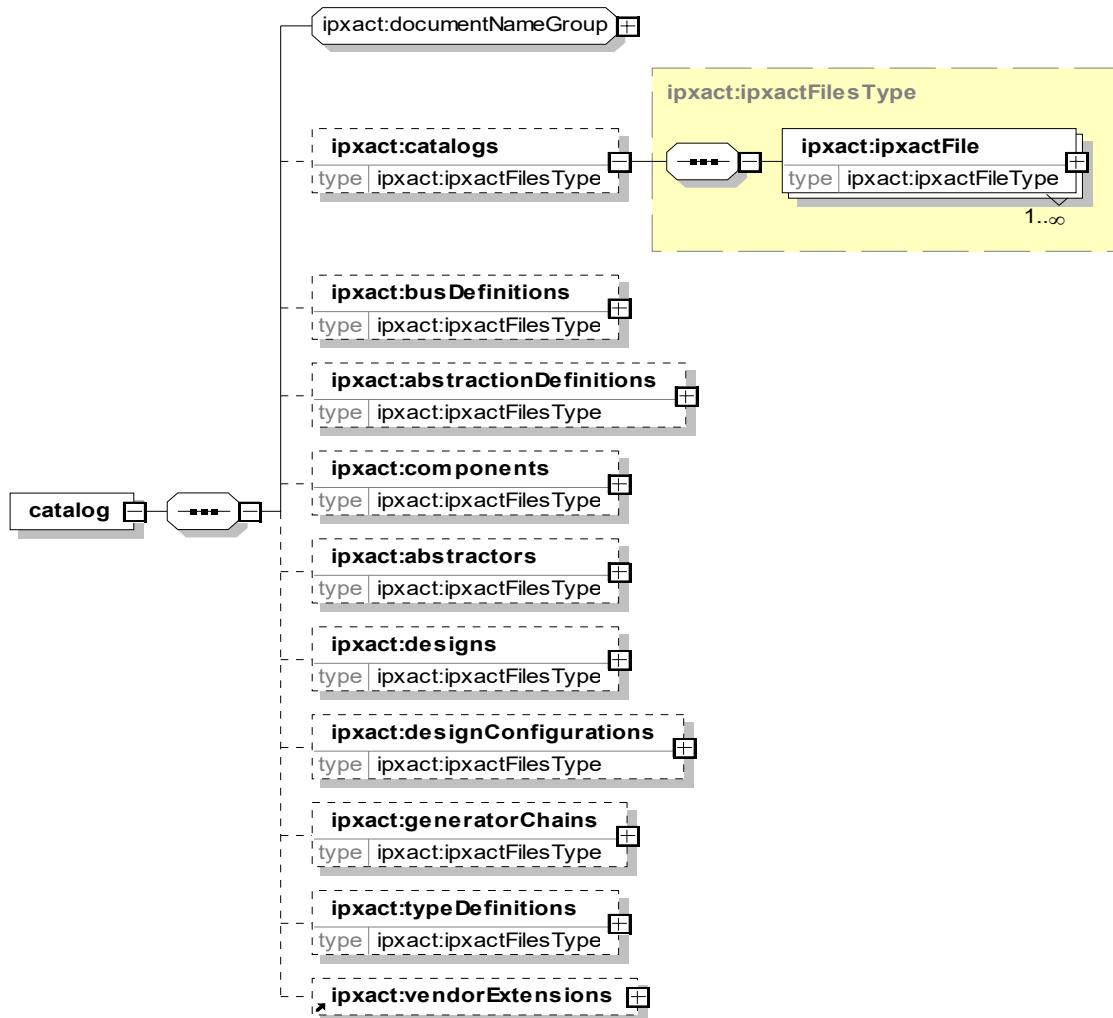
12.1 catalog

Systems described with IP-XACT can consist of a single XML file, but more often reference a large number of files (e.g., components, bus/abstraction definitions). The top-level **catalog** element provides a mechanism for documenting the locations of these files and the top-level details (VLDN and document type) giving design environments a standard means of locating IP-XACT documents.

A *catalog* can document the location, VLDN, and file type of any number of IP-XACT documents, including other catalogs.

12.1.1 Schema

The following schema details the information contained in the **catalog** element, which contains the optional list elements **catalogs**, **busDefinitions**, **abstractionDefinitions**, **components**, **abstractors**, **designs**, **designConfigurations**, **generatorChains**, and **typeDefinitions**.



12.1.2 Description

The top-level **catalog** element references top-level IP-XACT elements and specifies where each can be found. The **catalog** contains the following elements and attributes:

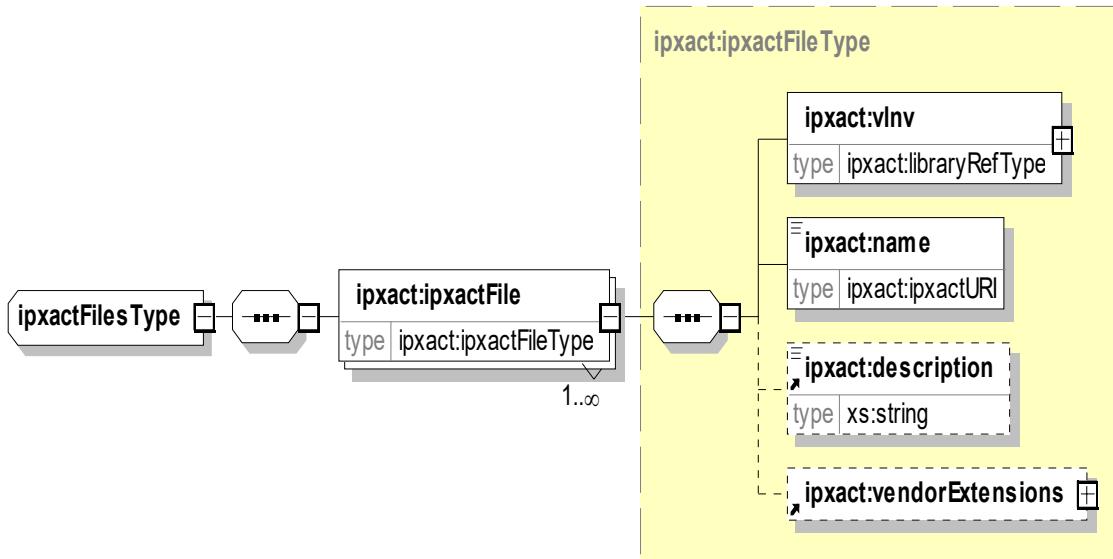
- a) **documentNameGroup** contains **versionedIdentifier**, **displayName**, **shortDescription**, and **description**. See [C.11](#).
- b) **catalogs** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **catalog** documents. See [12.2](#).
- c) **busDefinitions** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **busDefinitions** documents. See [12.2](#).
- d) **abstractionDefinitions** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **abstractionDefinitions** documents. See [12.2](#).
- e) **components** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **components** documents. See [12.2](#).
- f) **abstractors** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **abstractors** documents. See [12.2](#).
- g) **designs** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **designs** documents. See [12.2](#).
- h) **designConfigurations** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **designConfigurations** documents. See [12.2](#).
- i) **generatorChains** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **generatorChains** documents. See [12.2](#).
- j) **typeDefinitions** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **typeDefinitions** documents. See [12.2](#).
- k) **vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See [C.27](#).

See also [SCR 1.3](#), [SCR 1.10](#), [SCR 1.12](#), and [SCR 6.18](#).

12.2 ipxactFile

12.2.1 Schema

The following schema details the information contained in the **ipxactFile** element, which documents the existence of a top-level IP-XACT file with the indicated location and VLNV. The type of the IP-XACT file is expected to match that of the container element of the **ipxactFile**.



12.2.2 Description

The **ipxactFile** contains the following elements and attributes:

- vlnv** (mandatory; type: *libraryRefType*, see [C.18](#)) specifies the VLNV of the IP-XACT file referenced by the **name** element.
- name** (mandatory; type: *ipxactURI*, see [D.11](#)) identifies the containing element.
- description** (optional; type: *string*) allows a textual description of the containing element.
- vendorExtensions** (optional) adds any extra vendor-specific data related to the design configuration. See [C.27](#).

See also [SCR 1.15](#).

13. Addressing

This clause describes how addresses are transformed between a target's memory map and a initiator's address space. [Clause 14](#) also describes how to determine which bits of the memory map are visible in the initiator's address space.

Indirect interfaces and the associated indirectly accessible memory map do not directly add to the address map. Rather, each indirect interface requires a separate address map calculation. Address map calculation for indirect interfaces is the same as a target bus interface. The indirect address field and indirect data field are analogous to the logical address and data ports of a target bus interface.

13.1 Calculating the bit address of a bit in a memory map or address space

A *memory map* consists of a set of address blocks, subspace maps, and banks containing further address blocks, subspace maps, and banks (to any number of levels). To calculate the address of a bit within an address block or subspace map relative to the containing memory map, its bit address needs to be calculated relative to its parent. If that parent is a bank, first calculate how that bank modifies the address, and then continue working up the bank structure until the memory map is reached. An *address space* may contain a local memory map. The calculations for local memory maps and memory maps are identical.

A bit address in a memory map or address space is calculated by multiplying an address by the appropriate *addressUnitBits* value. A *registerFileDefinition*, *addressBlockDefinition*, *memoryMapDefinition*, and *memoryRemapDefinition* describe the *addressUnitBits* value for their subelements; if the *addressUnitBits* element is not present, then it is presumed to be 8. An *addressBlock*, *bank*, and *subspaceMap* described in a component memory map inherit the *addressUnitBits* value from that memory map. An *addressBlock* and *bank* described in a component address space local memory map inherit the *addressUnitBits* value from that address space.

- For a bit in a register, the bit address in its containing element is:

$$\text{register_bit_address_offset} = \text{register.addressOffset} \times \text{addressUnitBits} + \text{field.bitOffset} \quad (1)$$

- For a bit in a register file, the bit address in its containing element is:

$$\text{registerFile_bit_address_offset} = \text{registerFile.addressOffset} \times \text{addressUnitBits} + \text{register_bit_address_offset} \quad (2)$$

- For a bit in a nested register file of depth n :

$$\text{registerFile_bit_address_offset}(n) = \text{registerFile.addressOffset}(n) \times \text{addressUnitBits} + (\text{SUM } i=0: n-1: \text{registerFile_bit_address_offset}(i)) \quad (3)$$

- For a bit in an address block (where *bit_address_offset* can be a *register_bit_address_offset*, *registerFile_bit_address_offset*, or any other bit offset in the address block range), the bit address in its containing element is:

$$\text{addressBlock_bit_address_offset} = \text{addressBlock.baseAddress} \times \text{addressUnitBits} + \text{bit_address_offset} \quad (4)$$

- For a bit in a subspace map (where *bit_address_offset* is some bit offset in the range of the referenced address space or address space segment), the bit address in its containing element is:

$$\text{subspaceMap_bit_address_offset} = \text{subspaceMap.baseAddress} \times \text{memoryMap.addressUnitBits} + \text{bit_address_offset} \quad (5)$$

A bank may contain address blocks, subspace maps, and banks. For convenience, they are referred to as items. Each item has a width and a range.

If an item is an address block, then

$$item.width = addressBlock.width \quad (6)$$

$$item.bit_range = addressBlock.range \times addressUnitBits \quad (7)$$

If an item is a subspace map without a segment reference, then

$$item.width = subspaceMap.addressSpace.width \quad (8)$$

$$item.bit_range = subspaceMap.addressSpace.range \times addressSpace.addressUnitBits \quad (9)$$

If an item is a subspace map with a segment reference, then

$$item.width = subspaceMap.addressSpace.width \quad (10)$$

$$item.bit_range = subspaceMap.addressBlock.segment.range \times addressSpace.addressUnitBits \quad (11)$$

If an item is a serial bank containing n address blocks or subspace maps, then

$$item.width = (\text{MAX: } i=1: n: item_i.width) \quad (12)$$

$$item.bit_range = (\text{SUM: } i=1: n: item_i.width \times \text{ceil}((item_i.range \times item_i.addressUnitBits) / item.width)) \quad (13)$$

If an item is a parallel bank containing n address blocks or subspace maps, then

$$item.width = (\text{SUM: } i=1: n: item_i.width) \quad (14)$$

$$item.bit_range = item.width \times (\text{MAX: } i=1: n: \text{ceil}(item_i.range / item_i.width)) \quad (15)$$

For a bit in a serial bank (where $bit_address_offset$ is some bit offset in the n -th item in the bank), the bit address in its containing element is:

$$\begin{aligned} bank_bit_address_offset = & bank.baseAddress \times addressUnitBits + \\ & (\text{SUM: } i=1: n-1: item_i.bit_range) + bit_address_offset \end{aligned} \quad (16)$$

For a bit in a parallel bank (where $bit_address_offset$ is some bit offset in the n -th item), the bit address in its containing element is:

$$\begin{aligned} bank_bit_address_offset = & bank.baseAddress \times addressUnitBits + (\text{SUM: } i=1: n-1: item_i.width \times \\ & \text{ceil}(bit_address_offset / item_i.width)) + bit_address_offset \end{aligned} \quad (17)$$

For a bit in a nested bank (where $bit_address_offset$ is some bit offset in the serial or parallel bank), the bit address in its containing element is:

$$\begin{aligned} bank_bit_address_offset(n) = & bank.baseAddress(n) \times addressUnitBits + (\text{SUM } i=0: n-1: bank_bit_address_offset(i)) \end{aligned} \quad (18)$$

For an element in a (local) memory map (where $memoryMap_address_offset$ is some address offset in the memory map), the bit address is:

$$memoryMap_bit_address_offset = memoryMap_address_offset \times addressUnitBits \quad (19)$$

The content of a $memoryMap_bit_address_offset$ is defined by the contents of the bits in address blocks, subspace maps, and banks in the memory map and by the contents of address spaces that are bridged from the memory map using channels (see [13.4](#)) and transparent and opaque bridging (see [13.5](#)).

For an element in an address space (where $addressSpace_address_offset$ is some address offset in the address space), the bit address is:

$$addressSpace_bit_address_offset = addressSpace_address_offset \times addressUnitBits \quad (20)$$

The content of an *addressSpace_bit_address_offset* is defined by the content of the bits in a local memory map in the address space and by the contents of memory maps that are mapped into the address space using design interconnections.

The formulas above apply to elements without an array subelement. If an element has an array, then the formulas should be modified as follows where *index* is a vector in the array dimensions.

$$\begin{aligned} \text{field.bitOffset}(\text{index}) &= \\ \text{field.bitOffset} + \text{index} \times \max(\text{field.bitWidth}, \text{field.bitStride}) \end{aligned} \quad (21)$$

$$\begin{aligned} \text{register.addressOffset}(\text{index}) &= \\ \text{register.addressOffset} + \text{index} \times \max(k, \text{register.stride}) \times \text{addressUnitBits} \end{aligned} \quad (22)$$

where

k is the least integer such that $k \times \text{addressUnitBits} \geq \text{register.size}$

$$\begin{aligned} \text{registerFile.addressOffset}(\text{index}) &= \text{registerFile.addressOffset} + \text{index} \times \max(\text{registerFile.range}, \\ \text{registerFile.stride}) \times \text{addressUnitBits} \end{aligned} \quad (23)$$

$$\begin{aligned} \text{addressBlock.baseAddress}(\text{index}) &= \text{addressBlock.baseAddress} + \text{index} \times \max(\text{addressBlock.range}, \\ \text{addressBlock.stride}) \times \text{addressUnitBits} \end{aligned} \quad (24)$$

13.2 Calculating the bus address at the bus interface

For addressable initiator and target bus interfaces, i.e., bus interfaces that reference a bus definition that has **isAddressable** set to **true**, a bit address in an interface is calculated by multiplying an address by the appropriate **bitsInLau** value, which is described in that interface.

$$\text{target_bus_bit_address_offset} = \text{memory_map_bit_address_offset} \quad (25)$$

$$\text{initiator_bus_bit_address_offset} = \text{address_space_bit_address_offset} \quad (26)$$

$$\text{target_bus_address_offset} = \text{target_bus_bit_address_offset}/\text{target.bitsInLau} \quad (27)$$

$$\text{initiator_bus_address_offset} = \text{initiator_bus_bit_address_offset}/\text{initiator.bitsInLau} \quad (28)$$

$$\begin{aligned} \text{mirrored_target_bus_address_offset} &= \text{mirrored_target_bus_bit_address_offset}/ \\ \text{mirrored_target.bitsInLau} \end{aligned} \quad (29)$$

$$\begin{aligned} \text{mirrored_initiator_bus_address_offset} &= \text{mirrored_initiator_bus_bit_address_offset}/ \\ \text{mirrored_initiator.bitsInLau} \end{aligned} \quad (30)$$

Interfaces that do not explicitly reference a memory map or address space, e.g., hierarchical bus interfaces, have an implied memory map or address space.

13.3 Calculating the address at the indirect interface

The address of a bit at an indirect interface can be derived from the following equations:

$$\text{indirect_interface_bus_bit_address_offset} = \text{memory_map_bit_address_offset} \quad (31)$$

$$\begin{aligned} \text{indirect_interface_bus_address_offset} &= \\ \text{indirect_interface_bus_bit_address_offset}/\text{indirect_interface.bitsInLau} \end{aligned} \quad (32)$$

13.4 Address modifications of a channel

The address at the mirrored target interface can be derived from the following equation:

$$\text{mirrored_initiator_bit_address_offset} = \text{mirrored_target_bus_bit_address_offset} + \\ \text{mirroredTarget.baseAddress.remapAddress} \times \text{mirroredTarget.bitsInLau} \quad (33)$$

where

remapAddress is the remap address for the current mode of the channel

13.5 Address translation in a bridge

The address at the target interface for a bridge can be derived from the following equations:

a) For a transparent bridge:

$$\text{target_bit_address_offset} = \text{initiator_bit_address_offset} + \\ \text{initiator.addressSpaceRef.baseAddress} \times \text{addressSpace.addressUnitBits} \quad (34)$$

b) For an opaque bridge without an address space segment reference:

$$\text{target_bit_address_offset} = \text{initiator_bit_address_offset} + \\ \text{initiator.subspaceMap.baseAddress} \times \text{memoryMap.addressUnitBits} \quad (35)$$

c) For an opaque bridge with an address space segment reference:

$$\text{target_bit_address_offset} = \text{initiator_bit_address_offset} - \text{segment.addressOffset} \times \\ \text{addressSpace.addressUnitBits} + \text{initiator.subspaceMap.baseAddress} \times \text{memoryMap.addressUnitBits} \quad (36)$$

14. Data visibility

The addressing descriptions in [Clause 13](#) presume each bus interface maps only a single logical address port (a port with an **isAddress** qualifier) and a single logical data port (a port with an **isData** data qualifier). See also [5.6](#) and [5.9](#).

If a bus interface maps more than one address or data port, then each combination of address and data ports implies a separate addressing and data visibility calculation. To calculate the address map for a particular type of transaction, the data and address ports that transaction uses need to be known first.

The most common case for multiple data ports in a single bus interface is where there are separate read and write data ports; however, their relevant properties of the read and write data ports are typically identical—giving identical read and write address maps.

If an initiator or target interface blocks access to an address space or memory map, respectively, for specific modes using **modeRef** elements (see [6.7.5.2](#) and [6.7.6.2](#)), then data in that address space or memory map is not visible in those modes.

14.1 Mapped address bits mask

The mapped address bits need to be taken into account when calculating the data visibility to the interface by deriving a mask from the set of address bits mapped in the interface. This mask is 'bitwise anded' with the **bus_address**.

interface_mapped_address_bits = a mask derived from the set of address bits mapped in the interface (37)

interface_bus_address := target_bus_address | mirrored_target_bus_address | mirrored_initiator_bus_address | initiator_bus_address (38)

interface_visible_bus_address = interface_bus_address & interface_mapped_address_bits (39)

14.2 Address modifications of an interconnection

The bus address is carried between adjacent bus interfaces (target and mirrored target, initiator and mirrored initiator, or initiator and target) on the bus's **isAddress** logical port. If this port is a wire port, the address is always carried as parallel bits with the least significant bit of the address on logical bit **0** of the port. The interconnection can modify the address in two ways:

- a) If some address bits are not connected, addresses with those bits set are not accessible from the initiator.
 - 1) Examine the logical vectors in the port maps to determine which address bits are connected.
 - 2) Transactional ports always carry all address bits across the interconnection.
- b) If the value of **bitsInLau** differs on the two sides of the interconnection, the interpretation of the address as a bit address can vary by the ratio of the interfaces' **bitsInLau**. This, however, does not alter the actual bus address.

14.3 Bit steering in a channel

How addresses are modified within a channel depends on the value of **bitSteering** in the mirrored target interface. It also depends on the relative width of the mirrored initiator and mirrored target data ports, where this width is defined to be the total number of bits of the logical data port that are mapped in the bus interface. If **bitSteering** is 1, or the target is wider than or the same width as the initiator, the addresses

are simply modified to take into account any change in **bitsInLau** between the mirrored target and the mirrored initiator, as shown in the following formula:

$$\begin{aligned} \text{mirrored_initiator_steering_on_visible_bus_address} = \\ \text{mirrored_initiator_bus_address} \& \text{ mirrored_initiator_mapped_address_bits} \end{aligned} \quad (40)$$

If **bitSteering** is **0** and the mirrored target is narrower than the mirrored initiator, the address is adapted so all locations in the target's memory map are visible:

$$\text{mirroredInitiator_width} := \text{relative width of the dataport of the mirrored initiator interface} \quad (41)$$

$$\text{mirroredTarget_width} := \text{relative width of the dataport of the mirrored target interface} \quad (42)$$

$$\begin{aligned} \text{mirrored_initiator_bit_address} = \text{mirrored_target_bit_address} \bmod \text{mirroredTarget_width} \\ (\text{mirrored_target_bit_address} / \text{mirroredTarget_width}) \times \text{mirroredInitiator_width} \end{aligned} \quad (43)$$

$$\begin{aligned} \text{mirrored_initiator_steering_off_visible_bus_address} = \\ (\text{mirrored_initiator_bit_address} / \text{mirroredInitiator.bitsInLau}) \& \\ \text{mirrored_initiator_mapped_address_bits} \end{aligned} \quad (44)$$

Finally, **bitSteering** has a different meaning in a mirrored target interface from the meaning in an initiator or target interface. In an initiator or target interface, it means the component shall modify the bit lanes that are used for data when accessing narrow devices. In a mirrored target interface, it means the addresses from a mirrored initiator interface are not modified for transfers to a narrower mirrored target data port.

14.4 Visibility of bits

A bit in the target's memory map is visible in the initiator's address space if

- It is in an address range visible to the initiator.
- The initiator and target agree on the bit lane in which the bit should appear, and this bit lane is connected between the initiator and the target.

14.4.1 Visible address ranges

Two conditions need to be fulfilled for an address in the target to be visible to the initiator.

- a) The address at the mirrored target shall be within the range supported by the mirrored target interface:

$$\text{mirrored_target_visible_bus_address} < \text{mirroredTarget.baseAddress.range} \quad (45)$$

- b) The address in the address space shall be within the range supported by the initiator address space for that bus interface:

$$0 \leq \text{initiator_bus_bit_address} < \text{addressSpace.range} \times \text{addressSpace.addressUnitBits} \quad (46)$$

14.4.2 Bit lanes in memory maps

The local bit lane of a bit in an address block is

$$\text{address_block_bit_lane} = (\text{addressBlock.baseAddress} \times \text{addressUnitBits}) \bmod \text{addressBlock.width} \quad (47)$$

Similarly, in a subspace map the bit lane is

$$\text{subspace_map_bit_lane} = (\text{subspaceMap.baseAddress} \times \text{addressUnitBits}) \bmod \text{addressSpace.width} \quad (48)$$

where

addressSpace.width is the width of the address space of the referenced initiator bus interface

If the address block or subspace map is at the top level of the memory map or only within serial banks, the bit lane in the memory map is the local bit lane.

$$local_bit_lane := address_block_bit_lane \mid subspace_map_bit_lane \quad (49)$$

If it is item *n* in a parallel bank, then:

$$bank_bit_lane = (\text{SUM } i=0: n-1: item_i.width) + local_bit_lane \quad (50)$$

If it is in multiple hierarchical parallel banks, this formula is applied at each higher level with the lower-level *bank_bit_lane* replacing *local_bit_lane*.

The bit lane in the memory map is the top-level *bank_bit_lane*.

14.4.3 Bit lanes in address spaces

The bit lane in an address space can be derived from the following equations:

$$address_space_bit_address = initiator_bit_address \quad (51)$$

$$address_space_bit_lane = address_space_bit_address \bmod addressSpace.width \quad (52)$$

14.4.4 Bit lanes in bus interfaces

In a bus interface, the logical bit numbers of the data port carry the corresponding bit lanes. For example, if a target bus interface has a data port with a logical vector of [15:8], this port can access bit lanes 15 to 8 of the memory map and logical bit lanes 15 to 8 in the connected mirrored target or initiator interface.

14.4.5 Bit lanes in channels

All bus interfaces on a channel shall use the same logical numbering of data port bits. As a result, data bits cannot be moved between bit lanes in a channel by giving the mirrored bus interfaces different logical to physical mappings on their data ports.

14.4.6 Bit steering in initiators and targets

Bit steering takes effect only when the initiator and the target have data ports of different widths. If they do and bit steering is enabled (i.e., *bitSteering* is 1 in the initiator or target interface) for the bus interface with the wider data port, then this data port shall move its copy of output data to the correct bit lanes for the narrower port and read its input data from the correct bit lanes for the narrower port.

If bit steering is disabled in the wider port, the initiator can access data at a particular address only when the bit lane for that address in the address space is connected (through the bus interfaces and a channel) to the bit lane for the corresponding address in the memory map.

The following also apply:

- The **bitSteering** value has a different meaning in mirrored targets. See [13.2](#).
- Some buses with bit steering may support only certain data port widths. Describing which widths are supported is outside the scope of IP-XACT.

- Bit steering allows software or hardware away from the bus interface to work without knowing the width of devices on the far side of the bus. To provide this functionality, a bus supporting bit steering normally gives the same address bits to all devices, irrespective of their widths, and does not adapt addresses to the width of the target bus interfaces (i.e., `bitSteering` is `1` in the mirrored target bus interfaces). Thus, a non-bitsteering initiator on such a bus has access to only some of the memory rows of narrower targets.

Annex A

(informative)

Bibliography

Bibliographic references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

- [B1] IEC/IEEE 61691-1-1, Behavioral languages—Part 1: VHDL language reference manual.^{26, 27}
- [B2] IEEE Std 754™-1985, IEEE Standard for Binary Floating-Point Arithmetic.²⁸
- [B3] IEEE Std 1364™, IEEE Standard for Verilog Hardware Description Language.
- [B4] IEEE Std 1666™, IEEE Standard for SystemC Language Reference Manual.
- [B5] IETF RFC 2119, *Key words for Use in RFCs to Indicate Requirement Levels*, Bradner, S., Best Current Practice: 14.²⁹
- [B6] *International System of Units*.³⁰
- [B7] IP-XACT Leon Register Transfer and Transaction Level Examples.³¹
- [B8] IP-XACT Schema.³²
- [B9] IP-XACT TGI WSDL.³³

²⁶IEC publications are available from the International Electrotechnical Commission (<http://www.iec.ch/>). IEC publications are also available in the United States from the American National Standards Institute (<http://www.ansi.org/>).

²⁷IEEE publications are available from The Institute of Electrical and Electronics Engineers, Inc. (<http://standards.ieee.org/>).

²⁸The IEEE standards or products referred to in this annex are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

²⁹Available from <http://www.ietf.org/rfc/rfc2119.txt>.

³⁰Available at <http://physics.nist.gov/cuu/Units/>.

³¹Available from https://www.accellera.org/images/activities/committees/ip-xact/Leon2_1685-2022.zip

³²Available from <http://www.accellera.org/XMLSchema/IPXACT/1685-2022/>

³³Available from <http://www.accellera.org/XMLSchema/IPXACT/1685-2022/TGI/TGI.wsdl>

Annex B

(normative)

Semantic consistency rules

For an IP-XACT document or a set of IP-XACT documents to be valid, they shall, in addition to conforming to the IP-XACT schema, obey certain semantic rules. While many of these are described informally in other clauses of this document, this annex defines them formally. Tools generating IP-XACT documents shall ensure these rules are obeyed. Tools reading IP-XACT documents shall report any breaches of these rules to the user.

B.1 Semantic consistency rule definitions

The following definitions apply when determining a semantic consistency rule (SCR) interpretation.

B.1.1 Compatibility of busDefinitions

A set of **busDefinitions** are considered compatible if they are related by the **extends** relationship as described in [5.12.1](#). A **busDefinition** is always compatible with itself.

B.1.2 Interface mode of a bus interface

Specifies which of the following exclusive subelements of the **busInterface** is *present*: **initiator**, **target**, **system**, **mirroredInitiator**, **mirroredTarget**, **mirroredSystem**, or **monitor**. For instance, a **busInterface** containing a **system** subelement is considered to have an interface mode of **system**.

B.1.3 Compatibility of abstractionDefinitions

A set of **abstractionDefinitions** are considered compatible if they are related by the *extending* relationship as described in [5.12.2](#). Two configured **abstractionDefinitions** are compatible with each other if they meet the *extending* requirements.

B.1.4 Element referenced by configurableElementValue element

A **configurableElementValue** element is considered to reference a configurable element if the value of the **referenceId** attribute of the **configurableElementValue** matches the **parameterId** of that configurable element.

B.1.5 Memory mapping

If an access is not specified, the value defaults to the value from the level above. If the top level is not specified, the access defaults to read-write.

B.1.6 Port connection equivalence class

The *port connection equivalence class* of a (logical or component) port is the set of model and logical ports that can be reached from that port through any sequence of the following:

- a) Bus interfaces' logical-to-physical port maps
- b) Interconnections between logical ports implied by interconnections between bus interfaces using the same abstraction of the bus
- c) Ad hoc connections

B.1.7 Logical and physical ports

- a) If a **physicalPort** has a **partSelect** subelement, the physical bits specified by the **partSelect** are ordered using System Verilog bit ordering rules. If there is no **partSelect** subelement, the bit ordering is defined by the **arrays** subelements (if any) followed by the **vectors** subelements (if any) of the model port, again using System Verilog bit ordering rules. This bit ordering defines a linear order of the specified bits from `physical.left` to `physical.right`. If the model port does not have a range defined, then `physical.left = physical.right = 0`.
- b) If a **logicalPort** element has a **range** subelement, its range shall be `[left:right]`, where **left** and **right** are the left and right values of the **range** subelement. If a **logicalPort** element does not have a **range** subelement and the **physicalPort** element in the same **portMap** references a wire port, then its range shall be taken as `[abs(physical.left - physical.right) : 0]`.
- c) A logical bit of a port is mapped if it is included in the range of a **logicalPort** element referencing that bus interface port.

B.1.8 Addressable bus interface

A bus interface shall be *addressable* if the **isAddressable** element of the bus definition it references has the value **true**.

B.1.9 Field connection graph

The **field connection graph** of a given field or fieldDefinition in a given combination of modes is a directed graph. The given combination of modes determines the computed register field access (see [C.3](#)). The set of nodes contains a pair of nodes $m(f)$ (called mapping node) and $n(f)$ (called instance node) for each field or fieldDefinitions f of which the bit field values can be affected through the given field or fieldDefinition.

The set of arcs contains an arc from $m(f)$ to $n(f)$ if f has write access³⁴ in the given modes; an arc from $m(f)$ to $m(g)$ if write accesses to f result in write accesses to g in the given combination of modes; the write accesses to g may be the result of the fact that f broadcasts³⁵ to g , or of the fact that f and g are in the same register and that the bit offset of f is smaller than the bit offset of g . The set of arcs contain an arcs from $n(f)$ to $n(g)$ and from $n(g)$ to $n(f)$ if f is an alias of g .

Here are several examples to help clarify a field connection graph:

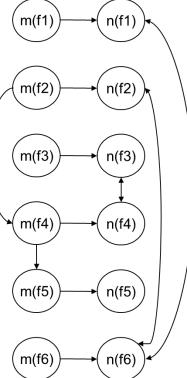
- a) Example 1 [Violation of [SCR 7.24](#)]: Consider a component with six fields named $f1$, $f2$, $f3$, $f4$, $f5$, and $f6$ with the following aliasing and broadcasting properties:
 - $f1$ is an alias of $f6$

³⁴write access is defined as computed register field write access as defined in [C.3](#).

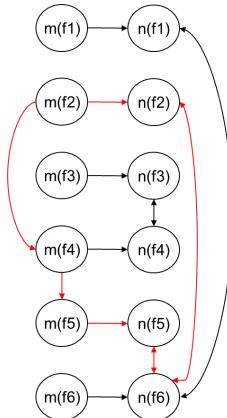
³⁵broadcasts element is fieldAccessPolicy modeRef specific.

- f2 is a broadcast to f4 and an alias of f6
- f3 is an alias of f4
- f4 is a broadcast to f5

Then the corresponding field connection graph can be depicted as:

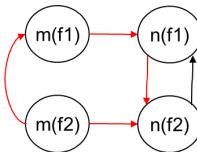


This graph satisfies [SCR 7.24](#) and [SCR 7.25](#). If f5 is also an alias of f6, then the graph contains an additional arc between n(f5) and n(f6). As a result it violates [SCR 7.24](#) because a write transaction to f2 will result in two drivers for f6 following the arcs indicated in red.



- b) Example 2 [Violation of [SCR 7.25](#)]: Consider a component with two fields named f1 and f2 with the following aliasing and broadcasting properties:
- f1 is an alias of f2
 - f2 is a broadcast to f1

Then the corresponding field connection graph can be depicted as:



This graph violates [SCR 7.25](#) because the arcs indicated in red form a cycle that contains two mapping nodes.

B.2 Rule listings

The semantic rules listed here can be checked purely by manually examining a set of IP-XACT documents. In [Table B.1](#) through [Table B.15](#), *Single doc check* indicates a rule can be checked purely by manually examining a single IP-XACT document. Rules for which *Single doc check* is No require the examination of the relationships between IP-XACT documents. When *Post config* is Yes, that rule applies only after configuration has been completed.

NOTE—Where these tables contain references to the values of elements and those elements are configurable in IP-XACT, the values used are the configured values (not the XML element values).

B.2.1 Cross-references and VLNVs

Table B.1—Cross-references and VLNVs

Name	Rule	Single doc check	Post config
SCR 1.1 <i>uniqueVLNV</i>	Every IP-XACT document visible to a tool shall have a unique VLVN. Applies only to documents visible to a particular tool or DE at one time. In particular, users are likely to store multiple versions of the same documents, with the same VLNVs, in source control systems. See also C.28.2 and C.28.4 .	No	No
SCR 1.2 <i>anyVLNVRefMustExist</i>	Any VLVN in an IP-XACT document used to reference another IP-XACT document shall precisely match the identifying VLVN of an existing IP-XACT document. In the schema, such references always use the attribute group <i>versionedIdentifier</i> . See also C.28.2 and C.28.4 .	No	No
SCR 1.3 <i>busDefExtendsVLNVIsBusDef</i>	The VLVN in an extends element in a bus definition shall be a reference to a bus definition. See also 5.2.2 .	No	No
SCR 1.4 <i>busTypeVLNVIsBusDef</i>	The VLVN in a busType element in a bus interface or abstraction definition shall be a reference to a bus definition. See also 6.7.1 .	No	No
SCR 1.5 <i>designRefVLNVIsDesign</i>	The VLVN in a designRef element in a design configuration or designInstantiation element in a component shall be a reference to a design . See also 11.2.2 .	No	No
SCR 1.6 <i>designCfgRefVLNVIsDesignCfg</i>	The VLVN in a designConfigurationRef element in a designConfigurationInstantiation element in a component shall be a reference to a designConfiguration .	No	No
SCR 1.7 <i>cfgChainRefVLNVIsGenChain</i>	The VLVN in a generatorChainRef element in a design configuration shall be a reference to a generator chain. See also 11.2.2 and 11.3 .	No	No
SCR 1.8 <i>selChainRefVLNVIsGenChain</i>	The VLVN in a generatorChainRef subelement of the element generatorChainSelector in a generator chain shall be a reference to a generator chain. See also 10.2.2 .	No	No
SCR 1.9 <i>compRefVLNVIsComp</i>	The VLVN in a componentInstanceRef element in a design shall be a reference to a component . See also 7.2.2 .	No	No
SCR 1.10 <i>legalTopDocTypes</i>	The XML document element of an IP-XACT document shall be an abstractor , abstractionDefinition , busDefinition , component , design , designConfiguration , generatorChain , typeDefinitions , or catalog element. See also 5.2.2 , 5.3 , 6.1.2 , 7.1.2 , 8.1.2 , 10.1.2 , and 11.2.2 .	Yes	No

Table B.1—Cross-references and VLNVs (continued)

Name	Rule	Single doc check	Post config
SCR 1.11 <i>absTypeVLNVIsAbsDef</i>	The VLVN in an abstractionType element in a component or abstractor shall reference an abstractionDefinition . See also 8.1.2 .	No	No
SCR 1.12 <i>busTypeMustMatch</i>	If a bus interface contains an abstractionType subelement, the abstraction definition's busType element and the bus interface's busType element shall reference the same bus definition. In other words, the abstraction referenced shall be an abstraction of the referenced bus. See also 5.2.2 , 5.3.2 , and 6.7.1.2 .	No	No
SCR 1.13 <i>absRefVLNVIsAbstractor</i>	The VLVN in an abstractorRef in a designConfiguration shall reference an abstractor . See also 11.3.2 .	No	No
SCR 1.14 <i>absDefExtendsVLNVIsAbsDef</i>	The VLVN in an extends element in an abstraction definition shall be a reference to an abstraction definition. See also 5.3.2 .	No	No
SCR 1.15 <i>CatalogVLNVReference</i>	The VLVN within an ipxactFile/vlnv element in a catalog shall refer to an IP-XACT file whose file type matches the container of the ipxactFile/vlnv element. For example, all VLNVs within the catalog busDefinitions element shall refer to IP-XACT busDefinitions elements. See also 12.2.2 .	No	No
SCR 1.16 <i>viewDesignConfigurationInstantiationReference</i>	If a view contains a designConfigurationInstantiation-Ref and does not also contain a designInstantiationRef , then the referenced design configuration document shall contain a designRef element. See also 6.15.1.2 and 11.2.2 .	No	Yes
SCR 1.17 <i>DesignRefsMatch</i>	If a view contains a designConfigurationInstantiation-Ref and a designInstantiationRef and the design configuration contains a designRef , then the referenced design VLNVs shall match. See also 6.15.1.2 and 11.2.2 .	No	Yes
SCR 1.18 <i>typeDefinitionsRefVLNVIsTypeDefinition</i>	The VLVN in a typeDefinitionsRef element in a component or typeDefinition shall be a reference to a typeDefinitions element. See also 6.2.2 .	No	No
SCR 1.19 <i>externalModeReference</i>	The modeRef attribute in an externalModeReference element shall reference a mode that exists in the typeDefinition referenced in the typeDefinitionsRef element. See also 6.2.2 .	No	No
SCR 1.20 <i>modeReference</i>	The modeRef attribute in a modeReference element shall reference a mode that exists in the encapsulating component or typeDefinitions . See also 6.2.2 .	Yes	No
SCR 1.21 <i>externalResetTypeReference</i>	The resetTypeRef attribute in an externalResetTypeReference element shall reference a resetType that exists in the typeDefinitions referenced in the typeDefinitionsRef element. See also 6.2.2 .	No	No
SCR 1.22 <i>resetTypeReference</i>	The resetTypeRef attribute in a resetTypeReference element shall reference a resetType that exists in the encapsulating component or typeDefinitions . See also 6.2.2 .	Yes	No
SCR 1.23 <i>externalViewReference</i>	The viewRef attribute in an externalViewReference element shall reference a view that exists in the typeDefinitions referenced in the typeDefinitionsRef element. See also 6.2.2 .	No	No

Table B.1—Cross-references and VLNVs (continued)

Name	Rule	Single doc check	Post config
SCR 1.24 <i>viewReference</i>	The viewRef attribute in a viewReference element shall reference a view that exists in the encapsulating component or typeDefinitions . See also 6.2.2 .	Yes	No
SCR 1.25 <i>fieldAccessPolicyDefinitionNameRefMustExist</i>	A fieldAccessPolicyDefinitionRef shall reference a name of a fieldAccessPolicyDefinition in the referenced typeDefinitions . See also 6.14.9 .	No	No
SCR 1.26 <i>enumerationDefinitionNameRefMustExist</i>	An enumerationDefinitionRef shall reference a name of an enumerationDefinition in the referenced typeDefinitions . See also 6.14.12 .	No	No
SCR 1.27 <i>fieldDefinitionNameRefMustExist</i>	A fieldDefinitionRef shall reference a name of a fieldDefinition in the referenced typeDefinitions . See also 6.14.8 .	No	No
SCR 1.28 <i>registerDefinitionNameRefMustExist</i>	A registerDefinitionRef shall reference a name of a registerDefinition in the referenced typeDefinitions . See also 6.14.2 .	No	No
SCR 1.29 <i>registerFileDefinitionNameRefMustExist</i>	A registerFileDefinitionRef shall reference a name of a registerFileDefinition in the referenced typeDefinitions . See also 6.14.6 .	No	No
SCR 1.30 <i>addressBlockDefinitionNameRefMustExist</i>	An addressBlockDefinitionRef shall reference a name of an addressBlockDefinition in the referenced typeDefinitions . See also 6.12.2 .	No	No
SCR 1.31 <i>memoryMapDefinitionNameRefMustExist</i>	A memoryMapDefinitionRef shall reference a name of a memoryMapDefinition in the referenced typeDefinition . See also 6.12.1 .	No	No
SCR 1.32 <i>memoryRemapDefinitionNameRefMustExist</i>	A remapDefinitionRef shall reference a name of a memoryRemapDefinition in the referenced typeDefinition . See also 6.13.1 .	No	No
SCR 1.33 <i>modesMustBeLinked</i>	All modeRef element values in an element that is referenced in an fieldDefinitionRef , registerDefinitionRef , registerFileDefinitionRef , addressBlockDefinitionRef , bankDefinitionRef , memoryMapDefinitionRef , or memoryRemapDefinitionRef shall occur in externalModeReference element values of the corresponding externalTypeDefinitions element. See also 6.2.2 , 6.12.1 , 6.12.2 , 6.13.1 , 6.14.2 , 6.14.6 , and 6.14.8 .	No	No
SCR 1.34 <i>resetTypesMustBeLinked</i>	All resetTypeRef element values in an element that is referenced in an fieldDefinitionRef , registerDefinitionRef , registerFileDefinitionRef , addressBlockDefinitionRef , bankDefinitionRef , memoryMapDefinitionRef , or memoryRemapDefinitionRef shall occur in externalResetTypeReference element values of the corresponding externalTypeDefinitions element. See also 6.2.2 , 6.12.1 , 6.12.2 , 6.13.1 , 6.14.2 , 6.14.6 , and 6.14.8 .	No	No

Table B.1—Cross-references and VLNVs (continued)

Name	Rule	Single doc check	Post config
SCR 1.35 <i>viewsMustBeLinked</i>	All view element values in an element that is referenced in an registerDefinitionRef , registerFileDefinitionRef , addressBlockDefinitionRef , bankDefinitionRef , memoryMapDefinitionRef , or memoryRemapDefinitionRef shall occur in externalViewReference element values of the corresponding externalTypeDefinitions element. See also 6.2.2 , 6.12.1 , 6.12.2 , 6.13.1 , 6.14.2 , and 6.14.6 .	No	No
SCR 1.36 <i>fieldSliceReference</i>	A fieldSlice in a fieldMap or mode shall reference an existing slice of a field in the encapsulating component . See also 6.10.2 and 6.15.7 .	No	Yes
SCR 1.37 <i>portSliceReference</i>	A portSlice in a fieldMap or mode shall reference an existing slice of a port in the encapsulating component . See also 6.10.2 .	No	Yes
SCR 1.38 <i>PowerDomainIntRef</i>	In a powerDomainLink , the externalPowerDomainReference attribute shall reference a powerDomain defined on the component referenced by this instance. See also 7.2.2 .	No	No
SCR 1.39 <i>PowerDomainExtRef</i>	In a powerDomainLink , the externalPowerDomainRef attribute (resolved expression) shall reference a powerDomain defined on the component referencing this design (or referencing a designConfig that references this design). See also 7.2.2 .	No	No
SCR 1.40 <i>fieldMapNoAlias</i>	A fieldMap shall not reference a field that is an alias of another field. See also 6.14.8.2 and 6.15.7.2 .	No	No
SCR 1.41 <i>NoPowDomRefInAbsDefPort</i>	An abstraction definition port shall not contain an isPowerEn qualifier with a powerDomainRef attribute. See also 5.6.2 .	Yes	No
SCR 1.42 <i>nonCircularVlNVReferences</i>	VlNV references in a design hierarchy shall not lead to circular referencing. See also C.9 .	No	Yes
SCR 1.43 <i>fieldReferenceArrayIndices</i>	In a fieldReferenceGroup , an indices element shall contain an index for each array dimension or shall not be present to reference the whole array. The total number of field elements referenced in the fieldReferenceGroup shall match the total number of field elements in the referencing field such that both can be linearized and the elements of the referencing and referenced field can be paired by the linear index value.	No	No

B.2.2 Interconnections

Table B.2—Interconnections

Name	Rule	Single doc check	Post config
SCR 2.1 <i>connectedIntfsMustExist</i>	In the attributes of an activeInterface , monitoredActiveInterface , or monitorInterface element, the value of the busRef attribute shall be the name of a busInterface in the component description referenced by the VLVN of the component instance named in componentInstanceRef and optional path attributes. See also 7.3.1.2 and 7.4.2 .	No	No
SCR 2.2 <i>connectedIntfsCompat</i>	In the subelements of an interconnection , the bus interfaces referenced by all activeInterface and hierInterface subelements shall be compatible, i.e., the busType elements within the busInterface elements shall reference compatible busDefinitions . See also 6.5.1 , 6.5.2 , 6.5.3 , and 7.3.1.2 .	No	Yes
SCR 2.3 <i>intfConnectedOnlyOnce</i>	A particular component/bus interface combination shall appear in only one interconnection element in a design. See also 7.3.1.2 .	Yes	Yes
SCR 2.4 <i>activeMstConnect</i>	An active interface of type initiator shall connect only to active interfaces of type target or mirrored-initiator or hierarchical interfaces of type initiator . See also 7.3.1.2 .	No	No
SCR 2.5 <i>activeMSlvConnect</i>	An active interface of type mirrored-target shall connect only to active interfaces of type target or hierarchical interfaces of type mirrored-target . See also 7.3.1.2 .	No	No
SCR 2.6 <i>hierSlvConnect</i>	A hierarchical interface of type target shall connect only to active interfaces of type target . See also 7.3.1.2 .	No	No
SCR 2.7 <i>hierMMstConnect</i>	A hierarchical interface of type mirrored-initiator shall connect only to active interfaces of type mirrored-initiator . See also 7.3.1.2 .	No	No
SCR 2.8 <i>systemConnect</i>	An active interface of type system shall connect only to active interfaces of type mirrored-system or hierarchical interfaces of type system . See also 7.3.1.2 .	No	Yes
SCR 2.9 <i>MstSystemConnect</i>	An active interface of type mirrored-system shall connect only to active interfaces of type system or hierarchical interfaces of type mirrored-system . See also 7.3.1.2 .	No	Yes
SCR 2.10 <i>interconnectionDriver</i>	An interconnection element without system interfaces or mirrored-system interfaces shall contain one driving interface, which is an active interface referencing an initiator, a mirrored-target, or a hierarchical interface referencing a target or mirrored-initiator. See also 7.3.1.2 .	No	Yes
SCR 2.11 <i>MstToSlvBitsLAUMatch</i>	In a direct initiator-to-target connection, the value of bitsInLAU in the initiator's bus interface shall match the value of bitsInLAU in the target's bus interface. See also 6.5.1 and 7.3.1.2 .	No	No

Table B.2—Interconnections (continued)

Name	Rule	Single doc check	Post config
SCR 2.12 <i>MstToSlvIsDirectConnect</i>	In a direct initiator-to-target connection, the bus-Definitions referenced by the busInterfaces shall have a direct-Connection element with the value true . See also 6.5.1 and 7.3.1.2 .	No	No
SCR 2.13 <i>SysToMSysGroupsMatch</i>	In a connection between a system interface and a mirrored-system interface, the values of the group elements of the two bus interfaces shall be identical. See also 6.5.1 , 6.5.2 , 6.7.4.2 , and 7.3.1.2 .	No	No
SCR 2.14 <i>EndianessMustMatch</i>	The endianess in all bus interfaces shall match for any interconnection using an addressable bus. If the endianess is not specified at either bus interface, it is presumed to be little endian. See also 6.5.1 , 6.5.2 , 6.7.1.2 , and 7.3.1.2 .	No	No
SCR 2.15 <i>ConnectionRequired</i>	If a design contains a component with a busInterface that has a connectionRequired element with the value true , that busInterface shall be included in an interconnection of the design. See also 6.7.1.2 and 7.3.1.2 .	No	No
SCR 2.16 <i>MonIntfPathMustExist</i>	A monitorInterconnection with interfaces that contain a path attribute with a componentInstanceRef and busRef shall exist in all hierarchical views. See also 7.4 .	No	Yes
SCR 2.17 <i>broadcastConstraint</i>	An interconnection may not contain more than two total activeInterface and hierInterface elements unless the underlying bus definition has the broadcast element set to true . See also 5.2.2 and 7.3.1.2 .	Yes	Yes
SCR 2.18 <i>excludePortExists</i>	A physical port name referenced in an excludePort element shall match the name of a port defined in the ports list of the component . See also 7.4.2 .	No	No
SCR 2.19 <i>physicalPortExists</i>	The component abstractionType viewRef elements shall reference views such that all component ports referenced by physicalPort elements exist. See also 6.7.2.2 , 6.7.3.2 , 6.15.7.2 , and C.29 .	Yes	No

B.2.3 Channels, bridges, and abstractors

Table B.3—Channels, bridges, and abstractors

Name	Rule	Single doc check	Post config
SCR 3.1 <i>ChannelAbsDefsCompat</i>	Within a channel element, all the busInterfaceRef elements shall refer to compatible abstraction definitions, i.e., the VLNVs of the abstractionType elements within the busInterface elements shall reference compatible abstractionDefinitions . Compatibility of the abstraction definitions implies compatibility of their associated bus definitions. See also 5.3.2 and 6.9.2 .	No	Yes
SCR 3.2 <i>ChannelInfsMirrored</i>	All bus interfaces referenced by a channel shall be mirrored interfaces. See also 6.6.1 and 6.9.2 .	Yes	No
SCR 3.3 <i>MaxInitiatorsHonored</i>	A channel can be connected to no more mirrored-initiator busInterfaces than the least value of maxInitiators in the bus definitions referenced by the connected bus interfaces (whether these interfaces are mirrored-initiator or mirrored-target interfaces). A channel may connect ports with different bus definitions, and hence different values of maxInitiators , as long as the bus definitions are compatible. See also 6.9.2 .	No	Yes
SCR 3.4 <i>MaxTargetsHonored</i>	A channel can be connected to no more mirrored-target bus interfaces than the least value of maxTargets in the bus definitions referenced by the connected bus interfaces (whether these interfaces are mirrored-initiator or mirrored-target interfaces). A channel may connect ports with different bus definitions, and hence different values of maxTargets , as long as the bus definitions are compatible. See also 6.9.2 .	No	Yes
SCR 3.5 <i>BusIntfInOneChannelMax</i>	Each bus interface on a component shall connect to only one channel of that channel component. See also 6.9.2 .	Yes	Yes
SCR 3.6 <i>InitiatorRefIntfIsInitiator</i>	The interface referenced by initiatorRef subelement of a bridge element shall be an initiator . See also 6.7.6.2 .	Yes	No
SCR 3.7 <i>InterConnectionRefMustMatch</i>	The value of the interconnectionRef subelement of an interconnectionConfiguration element shall precisely match a design interconnection/name or a design monitorInterconnection/name of an interconnection described in the design referenced by the containing design configuration or referenced by the design in the designInstantiation referenced by the view referencing the containing design configuration. See also 11.3.2 .	No	No
SCR 3.8 <i>AbstractorModeMustBeInitiator</i>	An abstractors element of an interconnectionConfiguration element in a design configuration document that references an initiator-to-mirrored-initiator connection shall reference only abstractors with an abstractorMode of initiator . See also 11.3.2 .	No	No

Table B.3—Channels, bridges, and abstractors (continued)

Name	Rule	Single doc check	Post config
SCR 3.9 <i>AbstractorModeMustBeTarget</i>	An abstractors element of an interconnectionConfiguration element in a design configuration document that references a target-to-mirrored-target interconnection in the corresponding design shall reference only abstractors with an abstractorMode of target . See also 11.3.2 .	No	No
SCR 3.10 <i>AbstractorModeMustBeSystem</i>	An abstractors element of an interconnectionConfiguration element in a design configuration document that references a system-to-mirrored-system interconnection in the corresponding design shall reference only abstractors with an abstractorMode of system . See also 11.3.2 .	No	No
SCR 3.11 <i>AbstractModeMustBeDirect</i>	An abstractors element of an interconnectionConfiguration element in a design configuration document that references an initiator-to-target interconnection in the corresponding design shall reference only abstractors with an abstractorMode of direct . See also 11.3.2 .	No	No
SCR 3.12 <i>AbstractionChainStart</i>	In the list of abstractor elements within an abstractors element in an interconnectionConfiguration element, the first abstractionType element of the first referenced abstractor shall be compatible with the abstractionType element of the initiator, system, or mirrored-target endpoint of the interconnection. ^a See also 11.3.2 .	No	No
SCR 3.13 <i>AbstractionChainEnd</i>	In the list of abstractor elements within an abstractors element in an interconnectionConfiguration element, the second abstractionType element of the last referenced abstractor shall be compatible with the abstractionType element of the mirrored-initiator, mirrored-system, or target endpoint of the interconnection. ^a See also 11.3.2 .	No	No
SCR 3.14 <i>AbstractionChainMiddle</i>	In the list of abstractor elements within an abstractors element in an interconnectionConfiguration element, the first abstractionType element of every referenced abstractor, except the first, shall be compatible with the second abstractionType element of the previous abstractor in the interconnection-Configuration list. ^a See also 11.3.2 .	No	No
SCR 3.15 <i>AbstractionBusTypesMustMatch</i>	The VLNVs in the busType elements of both abstraction definitions referenced by an abstractor shall exactly match the VLVN in the busType element of the abstractor. See also 5.3.2 and 8.1.2 .	No	No
SCR 3.16 <i>AbstractionExtendsCondition</i>	If abstraction definition AA is an abstraction of bus definition A and abstraction definition AB is an abstraction of bus definition B, then abstraction definition AA shall contain an extends element referencing abstraction definition AB only if bus definition A contains an extends element referencing bus definition B. If abstraction definition AA extends abstraction definition AB, AA and AB need to be abstractions of different buses. See also 5.3.2 .	No	No

Table B.3—Channels, bridges, and abstractors (continued)

Name	Rule	Single doc check	Post config
SCR 3.17 <i>SubspaceInitiatorRefExists</i>	The interface referenced by the initiatorRef attribute of a subspaceMap element shall be an initiator interface. See also 6.12.9.2 .	Yes	No
SCR 3.18 <i>multiAbstractorBroadcast</i>	If multiple abstractors elements appear in an interconnectionConfiguration , then the referenced interconnection shall be a broadcast connection (i.e., contain more than two interfaces). See also 5.2.2 and 11.3.2 .	Yes	No

^a[SCR 3.12](#) – [SCR 3.14](#) mean the abstractors associated with an interconnection need to form a non-looping chain between the two ends.

B.2.4 Monitor interfaces and monitor interconnections

Table B.4—Monitor interfaces and monitor interconnections

Name	Rule	Single doc check	Post config
SCR 4.1 <i>ActiveInterfaceCondition</i>	An activeInterface or monitoredActiveInterface element shall reference an initiator , target , system , mirroredInitiator , mirroredTarget , or mirroredSystem interface. See also 6.5.3 , 7.3.1.2 , 7.4.2 , and 7.6.2 .	No	No
SCR 4.2 <i>MonitorInterfaceCondition</i>	The monitorInterface subelements of a monitorInterconnection element shall reference a monitor bus interface. See also 6.5.3 and 7.3.1.2 .	No	No
SCR 4.3 <i>MonitorModeMustMatch</i>	In a monitorInterconnection element, the value of the interfaceMode of the monitor interfaces shall match the mode of the monitoredActiveInterface . As a result, all the monitor interfaces shall have the same interface mode. See also 6.5.3 , 6.7.4.2 , and 7.3.1.2 .	No	No
SCR 4.4 <i>MonitorSystemGroupMatches</i>	A monitor interface shall be connected to a system or mirroredSystem interface only if it has a group subelement and the value of this element matches the value of the group subelement of the system or mirroredSystem interface. See also 6.5.3 , 6.7.4.2 , and 7.3.1.2 .	No	No
SCR 4.5 <i>InterfaceAppearsOnce</i>	A particular componentInstanceRef/busRef combination shall appear in only one monitorInterconnection element. This applies to both monitor and active interfaces; however, a single monitorInterconnection element can connect an active interface to many monitor interfaces. The same active interface can also appear in at most one interconnection element. See also 6.5.3 and 7.3.1.2 .	No	Yes
SCR 4.6 <i>MonitorPortDirRequirement</i>	All ports mapped in a busInterface with a mode of monitor shall have a direction of in for wire type ports or provides for transactional type ports. See also 6.5.3 .	Yes	No

B.2.5 Configurable elements

Table B.5—Configurable elements

Name	Rule	Single doc check	Post config
SCR 5.1 <i>expressionFieldValue</i>	The value of a field that allows expressions shall be an expression and shall reference only parameters in the document using their parameterId . See also C.6.2 .	Yes	No
SCR 5.2 <i>parameterIdRequired</i>	An parameterId attribute is required in any element with a resolve attribute value of user or generated . See also 6.15.6.2 and C.21.2 .	Yes	No
SCR 5.3 <i>componentInstanceConfigurableElementReferences</i>	configurableElementValue elements within componentInstance elements shall reference only configurable elements that exist in the component referenced by the enclosing componentInstance element, excluding those in a componentInstantiation or designConfigurationInstantiation . The value of the referenceId attribute of the configurableElementValue element shall match the value of the parameterId attribute of some configurable element of the component . See also 7.2.2 .	No	No
SCR 5.4 <i>configElementRefCondition</i>	configurableElementValue elements shall reference only configurable elements. See also C.9.2 .	No	No
SCR 5.5 <i>configurableElementMin</i>	If a configurableElementValue element references an element with a type attribute that does not specify a string and contains a minimum attribute, the value of the configurableElementValue element shall be greater or equal to the specified value of the minimum attribute. See also C.9.2 .	No	No
SCR 5.6 <i>configurableElementMax</i>	If a configurableElementValue element references an element with a type attribute that does not specify a string and contains a maximum attribute, the value of the configurableElementValue subelement shall be less than or equal to the specified value of the maximum attribute. See also C.9.2 .	No	No
SCR 5.7 <i>ConfigElementChoiceExists</i>	If a configurableElementValue element references an element with a choiceRef attribute, the value for configurableElementValue subelement shall be one of the values listed in the choice element referenced by the choiceRef attribute. See also C.8 and C.9.2 .	No	No
SCR 5.8 <i>designConfigurationInstantiationConfigurableElementReferences</i>	configurableElementValue elements within designConfigurationInstantiation elements shall reference only configurable elements that exist in the designConfiguration referenced by the enclosing designConfigurationInstantiation element; the value of the parameterId attribute of the configurableElementValue element shall match the value of the parameterId attribute of some configurable element of the design configuration. See also 6.15.1.2 and C.9.2 .	No	No

Table B.5—Configurable elements (continued)

Name	Rule	Single doc check	Post config
SCR 5.9 <i>viewConfigurationConfigurableElementReferences</i>	configurableElementValue elements within viewConfiguration elements shall reference only configurable elements that exist in the componentInstantiation or designConfigurationInstantiation referenced in the component view referenced by the enclosing view element; the value of the referenceId attribute of the configurableElementValue element shall match the value of the parameterId attribute of some configurable element of the component. See also 11.5.2 and C.9.2 .	No	No
SCR 5.10 <i>generatorChainConfigurableElementReferences</i>	configurableElementValue elements within generator-ChainConfiguration elements in design configuration documents elements shall reference only configurable elements that exist in the generator chain referenced by the generatorChainRef element. See also 11.2.2 and C.9.2 .	No	No
SCR 5.11 <i>abstractorConfigurableElementReferences</i>	configurableElementValue elements within interconnectionConfiguration elements shall reference only configurable elements that exist in the abstractor referenced by the enclosing abstractorRef element. See also 11.3.2 and C.9.2 .	No	No
SCR 5.12 <i>parameterImplicitCast</i>	A parameter's value or a configurable element's value shall be implicitly converter to the type and length specified by the type attribute and the vectors and arrays elements, respectively, resulting in an error when the value cannot be cast to the specified type., e.g., string to any other types and any other types to string . See also 7.2 .	No	No
SCR 5.13 <i>expressionsMinMax</i>	Expressions are bound by the values specified by the minimum and maximum attributes. See also C.7 .	Yes	No
SCR 5.14 <i>componentInstantiationParameterReferences</i>	Parameters inside a componentInstantiation cannot be referenced by expressions outside that componentInstantiation . See also 6.15.2.2 .	Yes	No
SCR 5.15 <i>designConfigurationInstantiationParameterReferences</i>	Parameters inside a designConfigurationInstantiation cannot be referenced by expressions outside that design-ConfigurationInstantiation . See also 6.15.2.2 .	Yes	No
SCR 5.16 <i>parameterInitializations</i>	The value of a parameter shall resolve to its indicated type. See also C.21 .	Yes	No
SCR 5.17 <i>expressionSyntax</i>	Expressions shall follow the SystemVerilog syntax. See also Annex E .	Yes	No
SCR 5.18 <i>expressionIDsExist</i>	Any ID used in an expression shall reference an existing parameterID . See also C.9.2 .	Yes	No
SCR 5.19 <i>expressionNonCircular</i>	Evaluation of an expression shall not lead to circular referencing.	Yes	No
SCR 5.20 <i>vectorDeclaration</i>	Vectors shall be specified only on parameters with a type of bit . See also C.21 .	Yes	No
SCR 5.21 <i>arrayDeclaration</i>	Array parameters shall be fully initialized; it shall be an error if the size of the array as determined by the default value differs from the size the array specified by the arrays elements. See also C.21 .	Yes	No

Table B.5—Configurable elements (continued)

Name	Rule	Single doc check	Post config
SCR 5.22 <i>arrayConfiguration</i>	Arrays shall be fully overridden when configured; it shall be an error if the size of the array resulting from the values specified in the configurable element differs from the size of the array specified by the arrays elements. See also C.21 .	Yes	No
SCR 5.23 <i>designInstantiationConfigurableElementReferences</i>	configurableElementValue elements within designInstantiation elements in component documents elements shall reference only configurable elements that exist in the design referenced by the enclosing designInstantiation element. See also 6.15.2.2 .	No	No
SCR 5.24 <i>busTypeConfigurableElementReferences</i>	configurableElementValue elements within busType elements shall reference only configurable elements that exist in the bus definition referenced by the enclosing busType element. See also 6.7.1.2 .	No	No
SCR 5.25 <i>abstractionTypeConfigurableElementReferences</i>	configurableElementValue elements within abstractionType elements shall reference only configurable elements that exist in the abstraction definition referenced by the enclosing abstractionType element. See also 6.7.1.2 .	No	No
SCR 5.26 <i>assertionValidity</i>	The value of an assertion shall evaluate to true . See also C.5 .	No	Yes
SCR 5.27 <i>typeDefinitionsRefConfigurableElementReferences</i>	configurableElementValue elements within typeDefinitionsRef elements shall reference only configurable elements that exist in the typeDefinitions referenced by the enclosing typeDefinitionsRef element. See also 6.2.2 .	No	No
SCR 5.28 <i>noRefIdToImmediate</i>	configurableElementValue attribute referenceId shall not reference configurable elements with a resolve attribute value of immediate . See also C.9 .	No	No

B.2.6 Ports

Table B.6—Ports

Name	Rule	Single doc check	Post config
SCR 6.1 <i>LogicalPortNameExists</i>	The value of the name subelement of any logicalPort element within an abstractionType element shall match the value of a logicalName element of the abstraction definition referenced by the abstractionType element. See also 6.7.3.2 .	No	No
SCR 6.2 <i>LogPortRequiresPortDir</i>	If the abstraction definition referenced by an abstraction-Type specifies an initiative value for a logical transactional port of requires for that interface mode of bus interface, the port map shall map that logical port only to a component port with an initiative value of requires , both , or phantom or to a component port with an allLogicalInitiativesAllowed attribute with the value true . For system interfaces, the port initiative values shall be looked up from the onSystem element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port initiative values shall be reversed before doing the comparison. See also 5.10.2 , 6.7.3.2 , and 6.15.19.2 .	No	No
SCR 6.3 <i>LogPortProvidesPortDir</i>	If the abstraction definition referenced by an abstraction-Type specifies an initiative value for a logical transactional port of provides for that interface mode of bus or abstractor interface, the port map shall map that logical port only to a component port with an initiative value of provides , both , or phantom or to a component port with an allLogicalInitiativesAllowed attribute with the value true . For system interfaces, the port initiative values shall be looked up from the onSystem element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port initiative values shall be reversed before doing the comparison. See also 5.10.2 , 6.7.3.2 , and 6.15.19.2 .	No	No
SCR 6.4 <i>LogPortBothPortDir</i>	If the abstraction definition referenced by an abstraction-Type specifies an initiative value for a logical transactional port of both for that interface mode of the bus or abstraction interface and if the bus interface has a port map, the port map shall map that logical port only to a component port with an initiative value of both or phantom or to a component port with an allLogicalInitiativesAllowed attribute with the value true . For system interfaces, the port initiative values shall be looked up from the onSystem element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port initiative values shall be reversed before doing the comparison. See also 5.10.2 , 6.7.3.2 , and 6.15.19.2 .	No	No

Table B.6—Ports (continued)

Name	Rule	Single doc check	Post config
SCR 6.5 <i>LogPortInPortDir</i>	<p>If the abstraction definition referenced by an abstraction-Type specifies a direction for a logical wire port of in for that interface mode of bus interface, the port map shall map that logical port only to a component port with a direction of in, inout, or phantom or to a component port with an allLogicalDirectionsAllowed attribute with the value true.</p> <p>For system interfaces, the port directions shall be looked up from the onSystem element with the group name matching that of the abstractionType's abstraction-definition.</p> <p>For mirrored interfaces, the bus port directions shall be reversed before doing the comparison.</p> <p>See also 5.7.2, 6.7.3.2, 6.15.8.2, and 8.6.2.</p>	No	No
SCR 6.6 <i>LogPortOutPortDir</i>	<p>If the abstraction definition referenced by an abstraction-Type specifies a direction for a logical wire port of out for that interface mode of bus interface, the port map shall map that logical port only to a component port with a direction of out, inout, or phantom or to a component port with an allLogicalDirectionsAllowed attribute with the value true.</p> <p>For system interfaces, the port directions shall be looked up from the onSystem element with the group name matching that of the abstractionType's abstraction-definition.</p> <p>For mirrored interfaces, the bus port directions shall be reversed before doing the comparison.</p> <p>See also 5.7.2, 6.7.3.2, 6.15.8.2, and 8.6.2.</p>	No	No
SCR 6.7 <i>LogPortInoutPortDir</i>	<p>If the abstraction definition referenced by an abstraction-Type specifies a direction for a logical wire port of inout for that interface mode of bus interface, the port map shall map that logical port only to a component port with a direction of inout or phantom or to a component port with an allLogicalDirectionsAllowed attribute with the value true.</p> <p>For system interfaces, the port directions shall be looked up from the onSystem element with the group name matching that of the abstractionType's abstraction-definition.</p> <p>For mirrored interfaces, the bus port directions shall be reversed before doing the comparison.</p> <p>See also 5.7.2, 6.7.3.2, 6.15.8.2, and 8.6.2.</p>	No	No
SCR 6.8 <i>LogPortPresence</i>	<p>If the abstraction definition referenced by an abstraction-Type specifies, for a port, a presence value of required for that interface mode of bus interface and if the bus interface has a port map, the port shall be in that port map.</p> <p>For system interfaces, the port presence shall be looked up from the onSystem element with the group name matching that of the abstractionType's abstraction-definition.</p> <p>Mirrored bus interfaces shall be looked up as if they were not mirrored.</p> <p>Port maps are optional, even on buses with required ports.</p> <p>See also SCR 6.16.</p> <p>The third possible presence value (optional) neither forces nor forbids the inclusion of the port in the port map.</p> <p>Presence does not apply to interfaces of type monitor.</p> <p>See also 5.10.2.</p>	No	No

Table B.6—Ports (continued)

Name	Rule	Single doc check	Post config
SCR 6.9 <i>OneWireDriver</i>	Only one component port in a port connection equivalence class may have the direction out , unless it is an analog port. See also 7.3.1.2 and 7.5.3 .	No	Yes
SCR 6.10 <i>OneTransactionalDriver</i>	Only one component port in a port connection equivalence class may have the initiative requires . See also 7.3.1.2 and 7.5.4 .	No	Yes
SCR 6.11 <i>ExtendedLogPortsExist</i>	If abstraction definition A extends abstraction definition B, then abstraction definition A shall have port elements for every port declared in abstraction definition B. If a port in abstraction definition B is not used in bus interfaces using abstraction definition A, then, in abstraction definition A, that port shall have a presence value of illegal for all bus interface modes. See also 5.3.2 and Table 2 .	No	Yes
SCR 6.12 <i>LogicalWireToPhysicalWire-OrStructured</i>	If the abstraction definition referenced by a bus or abstraction interface specifies a port is a wire port (i.e., the port element contains a wire subelement), the port map shall map that logical port only to a wire component port or a structured component (sub)port. If the physical port references a structured (sub)port then the (sub)port attribute packed value shall be true or the logical port shall not have a width. See also 5.5.2 , 6.7.3.2 , 6.15.8.2 , and 8.6.2 .	No	No
SCR 6.13 <i>LogicalTransToPhysicalTrans</i>	If the abstraction definition referenced by a bus or abstraction interface specifies a port is a transactional port (i.e., the port element contains a transactional subelement), the port map shall map that logical port only to a transactional component port. See also 5.9.2 , 6.7.3.2 , and 6.15.19.2 .	No	No
SCR 6.14 <i>LogPortSystemRefExists</i>	The value of the group subelement of an onSystem element shall match the value of one of the system group names referenced in the bus definition referenced by the abstraction definition containing the onSystem element. See also 5.5.2 and 5.9.2 .	No	No
SCR 6.15 <i>SystemGroupDefined</i>	The value of the group subelement of a system element shall match the value of one of the system group names referenced in the bus definition referenced by the bus interface containing the onSystem element. See also 6.7.4.2 .	No	No
SCR 6.16 <i>CantMapIllegalPresencePort</i>	If the abstraction definition defines ports with a presence value of illegal for a given interface mode, then the indicated ports may not appear in the port map of a bus interface of that mode type. For system interfaces, the port presence shall be looked up from the onSystem element with the group name matching that of the bus or abstraction interfaces. Mirrored bus and abstraction interfaces shall be looked up as if they were not mirrored. Port maps are optional, even on buses with required ports. See also SCR 6.8 . The third possible presence value (optional) neither forces nor forbids the inclusion of the port in the port map. See also 5.7.2 and 5.10.2 .	No	Yes

Table B.6—Ports (continued)

Name	Rule	Single doc check	Post config
SCR 6.17 <i>PhysicalPortRangeExists</i>	The range of a physicalPort shall be a subset of the range of the referenced port in the component's model element. See also 6.7.3.2 and B.1.6 .	Yes	No
SCR 6.18 <i>LogicalRangeMatchesPhysical</i>	Within any portMap , the sizes of the ranges of the physicalPort and the logicalPort shall be equal. See also 6.7.3.2 .	Yes	No
SCR 6.19 <i>LogicalRangeWithinDefinition</i>	If the abstraction definition port referenced by a logicalPort has a width defined, all elements in the range of the logical port shall be between <code>width-1</code> and 0. See also 6.7.3.2 .	No	No
SCR 6.20 <i>LogicalBitsMappedOnlyOnce</i>	Within a single bus interface, no logical bit may be mapped more than once, i.e., if two or more logicalPort elements for that bus interface reference the same abstraction definition port, their ranges shall not overlap. See also 6.7.3.2 .	Yes	Yes
SCR 6.21 <i>TransactionalPortBusConnection</i>	If a transactional port in a component is mapped in a bus interface to a transactional port in an abstraction definition, then the initiative , kind , busWidth , and protocolType elements in that component port shall match those defined in the abstraction definition's port. See also 6.7.3.1 and 6.15.19.2 .	No	No
SCR 6.22 <i>TransactionalPortConnection</i>	Transactional ports shall be connected together (by an ad hoc connection or through an interconnection) only if they have compatible initiative , busWidth , kind , and protocol elements and if none of them contains a transTypeDefs element or all of them contain compatible transTypeDefs elements. The following definitions apply here: <ol style="list-style-type: none"> Two initiatives with any provides-requires combination are compatible. Two kinds are compatible if either they are equal or both are sockets (<code>simple_socket</code>, <code>multi_socket</code>, or <code>tlm_socket</code>). Two busWidths are compatible if they are equal on both side of the connection. Two protocols are compatible if their protocolTypes are both <code>tlm</code>. Two protocols are compatible if their protocolTypes are both <code>custom</code> and (if defined) their payloads are the same. Two transTypeDefs are compatible if their typeParameters values are equal and their serviceTypeDefs typeNames (if defined and not implicit) are the same. See also 6.7.3.2 and 6.15.19.2 .	No	No
SCR 6.23 <i>Can'tDriveOutputPort</i>	A wire port with a direction of out shall not have a driver element. See also 6.15.12.2 .	Yes	Yes

Table B.6—Ports (continued)

Name	Rule	Single doc check	Post config
SCR 6.24 <i>AdHocPortWidthsMatch</i>	All wire (sub)ports and all structured (sub)ports with attribute packed set to true referenced in an ad hoc connection shall reference the same number of bits. If no range is specified for a non-scalar port, then the full range from the port definition is presumed. See also 7.5.3 .	No	No
SCR 6.25 <i>TiedValueDefaultHasDefault</i>	All ports references in an ad hoc connection that has a tied-Value of default shall have a default value defined. See also 7.5.2 .	No	No
SCR 6.26 <i>ViewlessComponentRestriction</i>	A component without views shall contain only phantom ports. See also 6.15.1.2 .	Yes	Yes
SCR 6.27 <i>VirtualComponentRestriction</i>	If isVirtual is true in a componentInstantiation element, then views referencing that componentInstantiation shall contain only phantom ports. See also 6.15.3.2 .	Yes	Yes
SCR 6.28 <i>StructuredDirectionMatch</i>	All values of all direction elements contained within wire , structured , and union elements within a structured element shall be equal. See also 6.15.23.2 .	Yes	No
SCR 6.29 <i>isIOPresence</i>	The isIO element shall only be present on a subPort contained within a structured element that has an interface element. See also 6.15.24.2 .	Yes	No
SCR 6.30 <i>PowerConstraintRangeRequirement</i>	The bits referenced by a powerConstraint range element must exist within the containing wire port. See also 6.15.8.2 .	Yes	No
SCR 6.31 <i>PowerConstraintOverlap</i>	The bits referenced by different powerConstraint range elements within the same port shall not overlap. See also 6.15.8.2 .	Yes	Yes
SCR 6.32 <i>AnalogDefaultValueCount</i>	The defaultValue within a driver element of an analog port defined as a vector shall be defined as a realVectorExpression and the number of elements of the default value should match the width of the port. See also 6.15.12.2 .	Yes	Yes
SCR 6.33 <i>matchPorts</i>	If the abstraction definition referenced by a bus or abstraction interface specifies, for a port, mapped in the bus-interface a match value of true the bus interface can only be connected to a bus-interface which maps this same port. See also 5.4.2 .	No	Yes
SCR 6.34 <i>allBits</i>	If an abstraction definition port has a width defined with allBits set, any bus interface containing a port map referencing that port shall map all the bits of that port; <i>i.e.</i> , <i>every bit in the range [width-1:0] shall be mapped precisely once in the port maps of that bus interface</i> . See also 5.7.2 .	No	Yes
SCR 6.35 <i>busDefinitionExtendsDirect-Connection</i>	If bus definition A extends bus definition B, then bus definition A cannot specify a different value for directConnection. See also 5.12.3 .	No	No

Table B.6—Ports (continued)

Name	Rule	Single doc check	Post config
SCR 6.36 <i>busDefinitionExtendsBroadcast</i>	If bus definition A extends bus definition B, then bus definition A cannot specify a different value for broadcast. See also 5.12.3 .	No	No
SCR 6.37 <i>busDefinitionExtendsIsAddressable</i>	If bus definition A extends bus definition B, then bus definition A cannot specify a different value for isAddressable. See also 5.12.3 .	No	No
SCR 6.38 <i>busDefinitionExtendsMaxInitiators</i>	If bus definition A extends bus definition B, then bus definition A cannot specify a higher number for maxInitiators. See also 5.12.3 .	No	Yes
SCR 6.39 <i>busDefinitionExtendsMaxTargets</i>	If bus definition A extends bus definition B, then bus definition A cannot specify a higher number for maxTargets. See also 5.12.3 .	No	Yes
SCR 6.40 <i>busDefinitionExtendsSystemGroupNames</i>	If bus definition A extends bus definition B, then bus definition A can only add new group names. See also 5.12.3 .	No	No
SCR 6.41 <i>abstractionDefinitionPortExtendsQualifiers</i>	If a port in abstraction definition A extends a port in abstraction definition B, then the extending port in abstraction definition A cannot specify different qualifiers. See also 5.12.3 .	No	No
SCR 6.42 <i>abstractionDefinitionPortExtendsDirection</i>	If a port in abstraction definition A extends a port in abstraction definition B, then the extending port in abstraction definition A cannot specify a different direction. See also 5.12.3 .	No	No
SCR 6.43 <i>abstractionDefinitionPortExtendsInitiative</i>	If a port in abstraction definition A extends a port in abstraction definition B, then the extending port in abstraction definition A cannot specify a different initiative. See also 5.12.3 .	No	No
SCR 6.44 <i>abstractionDefinitionPortExtendsKind</i>	If a port in abstraction definition A extends a port in abstraction definition B, then the extending port in abstraction definition A cannot specify a different kind. See also 5.12.3 .	No	No
SCR 6.45 <i>abstractionDefinitionPortExtendsBusWidth</i>	If a port in abstraction definition A extends a port in abstraction definition B, then the extending port in abstraction definition A cannot specify a different busWidth. See also 5.12.3 .	No	Yes
SCR 6.46 <i>abstractionDefinitionPortExtendsProtocol</i>	If a port in abstraction definition A extends a port in abstraction definition B, then the extending port in abstraction definition A cannot specify a different protocol. See also 5.12.3 .	No	No
SCR 6.47 <i>abstractionDefinitionPacket-FieldEarlierWidth</i>	If a packetField width expression uses <code>\$ipxact_packet-field_value()</code> , then the referenced packetField must be earlier in the packet. See also 5.11.2 .	Yes	No

Table B.6—Ports (continued)

Name	Rule	Single doc check	Post config
SCR 6.48 <i>abstractionDefinitionPacket-FieldValueFits</i>	If a packetField value is specified, then the expression must have the same number of bits as the width of that packetField. See also 5.11.2 .	Yes	No
SCR 6.49 <i>abstractionDefinitionPacket-FieldFixedOpcode</i>	If a packetField has the isOpcode qualifier, then it must have a fixed value. See also 5.11.2 .	Yes	No
SCR 6.50 <i>WireAndStructuredPortConnection</i>	Wire and structured ports shall be connected together (by an ad hoc connection or through an interconnection) only if they are compatible. The following definitions apply here: <ul style="list-style-type: none"> a) Two wire (subports) are compatible b) A wire (sub)port and a structured (sub)port are compatible if the structured port is packed c) Two structured (sub)ports are compatible if they both are packed d) Two structured (sub)ports are compatible if they both are unpacked and have the same type, i.e., <ul style="list-style-type: none"> 1) they are both a struct, or both a union, or both an interface, and 2) they both have the same typeName, typeDefinition, and typeParameters. See also 6.15.21.2 and 7.5.3 .	No	No
SCR 6.51 <i>constrainedRefArrayAndVector-Id</i>	A constrained attribute in a typeName element shall reference arrayId and vectorId attribute values of elements that are contained in the encapsulating port. See also 6.15.11.2 and 6.15.23.2 .	Yes	Yes
SCR 6.52 <i>structuredViewRefsAreConsistent</i>	viewRef elements in subPort wireTypeDef and struct-PortTypeDef elements shall reference the same views as the encapsulating structured port structPortTypeDef viewRef elements. See also 6.15.9.2 and 6.15.23.2 .	Yes	No
SCR 6.53 <i>isIoPortEquivalenceClass</i>	In a structured interface port connection equivalence class, all subPorts with attribute isIO set to true that are named the same, shall have the same port equivalence class. See also 6.15.24.2 and B.1.6 .	No	Yes
SCR 6.54 <i>structPortParamValuesAre-Equal</i>	All the configured structured interface port parameters defined on the component instances that contribute to a structured interface port connection equivalence class shall resolve to the same value. See also 6.15.25.2 and B.1.6 .	No	Yes
SCR 6.55 <i>partSelectIndices</i>	The indices of a partSelect element applied to a port reference shall select elements in the dimensions of that referenced port. See also C.22 .	No	Yes
SCR 6.56 <i>partSelectRangeOnLeaves</i>	In a reference to a structured port, explicit partSelect or implicit range element shall be applied only to the leave ports in the port reference. See also C.22 .	No	Yes

Table B.6—Ports (continued)

Name	Rule	Single doc check	Post config
SCR 6.57 <i>noInterfaceInStructOrUnion</i>	A structured port or subPort containing a struct or union element shall not have subPort elements that contain an interface element. See also 6.15.23.2 .	Yes	No
SCR 6.58 <i>structPortVectorRequiresPacked</i>	A structured port cannot contain a vector if the packed attribute is set to false. See also 6.15.23.2 .	Yes	No
SCR 6.59 <i>structPortInterfaceRequiresUnpacked</i>	A structured port cannot contain an interface element if the packed attribute is set to true. See also 6.15.23.2 .	Yes	No

B.2.7 Registers

Table B.7—Registers

Name	Rule	Single doc check	Post config
SCR 7.1 <i>RegisterOverlap</i>	No register shall have an addressOffset that falls within the address range of another register in the same address block, unless one of the registers and their alternateRegisters have non-conflicting computed register access value; see C.2 and C.3 . The address range of a register is the range [addressOffset, addressOffset + registerSize - 1] where registerSize is the size of the register in addressUnitBits. This is equal to $\dim[n-1] * \dim[n] * \dots * \dim[0] * \text{stride}$, where dim is the maximum number of elements for each of n dimensions. In other words, multiple readable and multiple writeable registers shall not overlap. See also 6.14.2.2 . Note that in this SCR, the least restrictive access values in the relevant accessPolicies and fieldAccessPolicies shall be used to determine the computed access values independent of the value of the mode conditions; see C.3 .	No	Yes
SCR 7.2 <i>BitOverlap</i>	No bit field shall have a bitOffset value that falls within the bit range of another bit field, unless one of the bit fields has computed access value read-only and the other has computed access value write-only , writeOnce , or no-access ; see C.2 and C.3 . The bit range of a bit field is the range [bitOffset, bitOffset + bitFieldWidth - 1] where bitFieldWidth is the size of a field in bits. This is equal to $\dim[n-1] * \dim[n] * \dots * \dim[0] * \text{bitStride}$, where dim is the maximum number of elements for each of n dimensions. In other words, multiple readable bit fields and multiple writeable bit fields shall not overlap. See also 6.14.2.2 and 6.14.8.2 . Note that in this SCR, the least restrictive access values in the relevant accessPolicies and fieldAccessPolicies shall be used to determine the computed access values independent of the value of the mode conditions; see C.3 .	No	Yes
SCR 7.3 <i>RegisterWithinBlock</i>	Any register in an address block shall fall entirely within that address block, i.e., for every $0 \leq \text{addressOffset} \leq \text{addressBlockRange} - \text{registerSize}$, where addressBlockRange is the range of the address block and registerSize is the size of the register in addressUnitBits. This is equal to $\dim[n-1] * \dim[n] * \dots * \dim[0] * \text{stride}$, where dim is the maximum number of elements for each of n dimensions. See also 6.14.2.2 .	No	Yes
SCR 7.4 <i>BitWithinRegister</i>	Any bit field in a register shall fall entirely within that register, i.e., for every $0 \leq \text{bitOffset} \leq \text{registerSize} - \text{bitFieldWidth}$, where registerSize is the size of the register in bits and bitFieldWidth is the width of a bit field. This is equal to $\dim[n-1] * \dim[n] * \dots * \dim[0] * \text{bitStride}$, where dim is the maximum number of elements for each of n dimensions. See also 6.14.2.2 and 6.14.8.2 .	No	Yes

Table B.7—Registers (continued)

Name	Rule	Single doc check	Post config
SCR 7.5 <i>RegisterSizeWithinBlock</i>	The size of any register shall be no greater than the width of the containing address block. See also 6.12.6.2 .	No	Yes
SCR 7.6 <i>RegisterWithinRegisterFile</i>	Any register in a register file shall fall entirely within that register file, i.e., for every register $0 \leq \text{register}.addressOffset \leq \text{registerFileRange} - \text{registerSize}$, where registerFileRange is the range of the register file and registerSize is the size of the register in addressUnitBits. This is equal to $\text{dim}[n-1] * \text{dim}[n] * \dots * \text{dim}[0] * \text{stride}$, where dim is the maximum number of elements for each of n dimensions. See also 6.14.2 , 6.14.3 , 6.14.6 , and 6.14.7 .	No	Yes
SCR 7.7 <i>RegisterFileWithinBlock</i>	Any register file in an address block shall fall entirely within that address block, i.e., for every register file $0 \leq \text{registerFile}.addressOffset \leq \text{addressBlockRange} - \text{registerFileSize}$, where addressBlockRange is the range of the address block and registerFileSize is the size of the register file in addressUnitBits. This is equal to $\text{dim}[n-1] * \text{dim}[n] * \dots * \text{dim}[0] * \text{stride}$, where dim is the maximum number of elements for each of n dimensions. See also 6.12.2 .	No	Yes
SCR 7.8 <i>BlockVolatileCondition</i>	volatile cannot be set to false for an addressBlock where any containing register or field already has volatile set to true . See also 6.14.2 , 6.14.3 , 6.14.8 , 6.14.9 , and 6.12.3 .	No	Yes
SCR 7.9 <i>RegisterVolatileCondition</i>	volatile cannot be set to false for a register where any containing field already has volatile set to true . See also 6.14.2 , 6.14.3 , 6.14.8 , and 6.14.9 .	No	Yes
SCR 7.10 <i>FieldUseEnumCondition</i>	When a field has writeValueConstraint/useEnumeratedValues set to true , it also shall have at least one enumeratedValue with the attribute usage set to write or read-write . See also 6.14.8 , 6.14.9 , and 6.14.12 .	Yes	Yes
SCR 7.11 <i>FieldConstraintRangeCondition</i>	When a field has a writeValueConstraint/minimum value and has a writeValueConstraint/maximum value, the value of maximum shall be greater than or equal to the value of minimum . See also 6.14.8 , 6.14.9 , and 6.14.12 .	Yes	Yes
SCR 7.12 <i>FieldTypeIdentifierCondition</i>	When multiple field elements have the same typeIdentifier , the field object shall contain the same contents for the elements in the fieldDefinitionGroup . For a field with an aliasOf element, the bitWidth , volatile , and resets of the field object is the bitWidth , volatile , and resets of the aliased field. See also 6.14.8 and 6.14.9 .	Yes	Yes
SCR 7.13 <i>RegisterTypeIdentifierCondition</i>	When multiple register or alternateRegister elements have the same typeIdentifier , the register object shall contain the same contents for the elements in the registerDefinitionGroup or alternateRegisterDefinitionGroup . For an alternateRegister element, the size of the register object is the size of the encapsulating register element. See also 6.14.3 and 6.14.5 .	Yes	Yes

Table B.7—Registers (continued)

Name	Rule	Single doc check	Post config
SCR 7.14 <i>RegisterFileTypeIdentifierCondition</i>	When multiple registerFile elements have the same typeIdentifier , the register file object shall contain the same contents for the elements in the registerFileDefinitionGroup . See also 6.14.6 and 6.14.7 .	Yes	Yes
SCR 7.15 <i>BlockTypeIdentifierCondition</i>	When multiple addressBlock elements have the same typeIdentifier , the address block object shall contain the same contents for the elements in the addressBlockDefinitionGroup . See also 6.12.3 .	Yes	Yes
SCR 7.16 <i>noWritePropsInROField</i>	A register field whose access type does not allow writing (access type read-only or no-access) shall not include any of the following subelements: modifiedWriteValue . See also 6.14.9.2 .	Yes	Yes
SCR 7.17 <i>noReadPropsInWOField</i>	A register field whose access type does not allow reading (access types of write-only, writeOnce or no-access) shall not include any of the following subelements: readAction . See also 6.14.9.2 .	Yes	Yes
SCR 7.18 <i>registerFileOverlap</i>	No register or register file shall have an addressOffset that falls within the address range of another register file in the same address block. The address range of a register file is the range $[addressOffset, addressOffset + registerFileSize - 1]$ where registerFileSize is the size of the register file in addressUnitBits . This is equal to $dim[n-1]*dim[n]*\dots*dim[0]*stride$, where dim is the maximum number of elements for each of n dimensions. See also 6.14.6 and 6.14.7 .	No	Yes
SCR 7.19 <i>resetTypeHARD</i>	A resetType with a name of HARD shall not be specified. See also 6.21 .	Yes	No
SCR 7.20 <i>resetTypeRefUnspecified</i>	Only one resetValue can be specified without a resetTypeRef (indicating the default/ HARD reset-type) on a field . See also 6.14.8.2 .	Yes	No
SCR 7.21 <i>aliasOfReference</i>	An aliasOf element shall reference another field in the same component , registerDefinition , registerFileDefinition , addressBlockDefinition , bankDefinition , memoryMapDefinition , or memoryReMapDefinition that does not have an aliasOf element. Note that a component and typeDefinition may be defined using multiple files due to referencing of external typeDefinitions files. See also 6.14.8.2 .	No	No
SCR 7.22 <i>broadcastToReference</i>	A broadcastTo element shall reference another field in the same component , registerDefinition , registerFileDefinition , addressBlockDefinition , bankDefinition , memoryMapDefinition , or memoryReMapDefinition . Note that a component and typeDefinitions may be defined using multiple files due to referencing of external typeDefinitions files. See also 6.14.9.2 .	No	No

Table B.7—Registers (continued)

Name	Rule	Single doc check	Post config
SCR 7.23 <i>uniqueBitFieldDriver</i>	A field access shall drive a bit field only once, i.e., a field or fieldDefinition field connection graph shall not contain an instance node that can be reached from more than one mapping node starting at a single mapping node. See also 6.14.8.2 and 6.14.9.2 . Note that in this SCR, the least restrictive access values in the relevant accessPolicies and fieldAccessPolicies shall be used to determine the computed access values independent of the value of the mode conditions; see C.3 .	No	No
SCR 7.24 <i>noFieldWriteCycle</i>	Field write accesses shall be single transactions, i.e., a field or fieldDefinition field connection graph shall not contain cycles containing more than one mapping node. See also 6.14.8.2 and 6.14.9.2 . Note that in this SCR, the least restrictive access values in the relevant accessPolicies and fieldAccessPolicies shall be used to determine the computed access values independent of the value of the mode conditions; see C.3 .	No	No
SCR 7.25 <i>EnumeratedValueWidth</i>	The width of value elements within an enumeratedValues list for a register field (whether specified explicitly or within an enumerationDefinition) must be less than or equal to the width of the containing register field. See also 6.14.12.2 and 9.3.2 .	No	No
SCR 7.26 <i>EnumeratedValueWithinWidth</i>	The enumeratedValue of an enumerationDefinition must fit within the enumerationDefinition width value. See also 6.14.12.2 and 9.3.2 .	No	No
SCR 7.27 <i>registerAlignment</i>	For each register in an addressBlock with attribute mis-alignmentAllowed set to false it shall hold that (register bit offset % addressBlock width) + register size <= addressBlock width. See also 6.12.2.2 and 6.12.6.2 .	No	No
SCR 7.28 <i>accessPolicyModeRefExists</i>	An accessPolicies element shall contain at most one accessPolicy element without modeRef elements. See also C.2 .	Yes	No
SCR 7.29 <i>fieldAccessPolicyModeRefExists</i>	A fieldAccessPolicies element shall contain at most one fieldAccessPolicy element without modeRef elements. See also 6.14.9 .	Yes	No
SCR 7.30 <i>accessRestrictionModeRefExists</i>	An accessRestrictions element shall contain at most one accessRestriction element without modeRef elements. See also 6.14.11 .	Yes	No
SCR 7.31 <i>RegisterFileWithinRegisterFile</i>	Any register file in a register file RF shall fall entirely within that register file RF, i.e., for every register file $0 \leq \text{registerFile.addressOffset} \leq \text{RF.range} - \text{registerFileSize}$, where RF.range is the range of the register file RF and registerFileSize is the size of the register file in addressUnitBits. This is equal to $\text{dim}[n-1] * \text{dim}[n] * \dots * \text{dim}[0] * \text{stride}$, where dim is the maximum number of elements for each of n dimensions. See also 6.14.6 and 6.14.7 .	No	Yes

B.2.8 Memory maps

Table B.8—Memory maps

Name	Rule	Single doc check	Post config
SCR 8.1 <i>BlockWidthCondition</i>	The width of an address block included in a memory map or memory map definition shall be a multiple of the memory map's addressUnitBits . See also 6.12.2.2 .	No	No
SCR 8.2 <i>NoSubspaceInParallelBank</i>	Neither a parallel bank nor banks within a parallel bank shall contain subspace maps. See also 6.12.5.2 , 6.12.7.2 , and 6.12.8.2 .	Yes	No
SCR 8.3 <i>addressBlockContent</i>	A register or register file cannot appear in an addressBlock with a usage of reserved. See 6.12.2 .	Yes	No
SCR 8.4 <i>BlockOverlap</i>	Two addressBlocks in the same memoryMap shall not overlap, i.e. no address block shall have a baseAddress that falls within the address range of another block in the same memoryMap . The address range of an address block is the range $[\text{baseAddress}, \text{baseAddress} + \text{addressBlockSize} - 1]$ where addressBlockSize is the size of the address block in addressUnitBits . This is equal to $\text{dim}[n-1] * \text{dim}[n] * \dots * \text{dim}[0] * \text{stride}$, where dim is the maximum number of elements for each of n dimensions. See also 6.12.2.2 .	No	Yes
SCR 8.5 <i>virtualRegisterContent</i>	The registerDefinitionGroup in a virtual register shall contain only typeIdentifier , size , and field elements. See also 6.12.4.2 .	Yes	No
SCR 8.6 <i>virtualAlternateRegisterContent</i>	When an alternate register is a child of a virtual register, the alternateRegisterDefinitionGroup shall contain only typeIdentifier and field elements. See also 6.14.4 .	Yes	No
SCR 8.7 <i>virtualFieldContent</i>	If a field is a child of a virtual register, the fieldData group shall contain only enumeratedValue and writeValueConstraint elements. See also 6.14.8 .	Yes	No
SCR 8.8 <i>StrideRangeRelationship</i>	The stride of an addressBlock or registerFile element shall be greater than or equal to the range of the address block or register file. See 6.12.2.2 and C.7.8 .	No	No
SCR 8.9 <i>StrideSizeRelationship</i>	The stride of a register element shall be greater than or equal to the size of the register. See 6.12.2.2 and C.7.8 .	No	No
SCR 8.10 <i>StrideBitwidthRelationship</i>	The bitStride of a registerField element shall be greater than or equal to the bitWidth of the field. See also 6.14.8 and C.7.8 .	Yes	No

B.2.9 Addressing

Table B.9—Addressing

Name	Rule	Single doc check	Post config
SCR 9.1 <i>AddressableInitiatorHasRef</i>	A non-hierarchical addressable initiator bus interface shall have an addressSpaceRef subelement. See also 6.7.5.2 .	Yes	No
SCR 9.2 <i>AddressableTargetHasMapOr-Bridge</i>	A non-hierarchical addressable target bus interface shall have a memoryMapRef subelement or one or more bridge subelements referencing addressable initiator bus interfaces. See also 6.7.6.2 .	Yes	No
SCR 9.3 <i>BitSteeringRestriction</i>	bitSteering is not allowed in mirrored-initiators, system, or mirrored-system interface modes. See also 6.7.1.2 .	Yes	No
SCR 9.4 <i>ChannelDataWidthRestriction</i>	Data widths in a channel shall all be a power 2 multiple of their bitsInLau . See also 6.7.1.2 .	Yes	No
SCR 9.5 <i>BitsInLauRestriction</i>	bitsInLau in a channel shall all be a power 2 multiple of the smallest bitsInLau . See also 6.7.1.2 .	Yes	No
SCR 9.6 <i>SegmentCondition</i>	For each segment within an addressSpace , everything between offsetAddress and offsetAddress + range-1 shall be contained within the range of that addressSpace . See also 6.11.1.2 and 6.11.2.2 .	Yes	No
SCR 9.7 <i>SegmentRefExists</i>	The segmentRef shall reference an existing segment of the addressSpace in the initiator referenced by the initiator-Ref . See also 6.12.9.2 .	Yes	No
SCR 9.8 <i>indirectDataField</i>	The field referenced as indirectData shall not be part of the memory-map referenced by the memoryMapRef . See also 6.8.2 .	Yes	No
SCR 9.9 <i>indirectAddressRefField</i>	The field referenced as indirectAddress shall not be part of the memory-map referenced by the memoryMapRef . See also 6.8.2 .	Yes	No
SCR 9.10 <i>indirectAddressRefFieldAccess</i>	The field referenced as indirectAddress shall not have an access-type of read-only , read-writeOnce , writeOnce , and no-access . See also 6.8.2 .	Yes	No

B.2.10 Hierarchy

Table B.10—Hierarchy

Name	Rule	Single doc check	Post config
SCR 10.1 <i>HierFamilyBusIntfBusTypes-Match</i>	All members of a hierarchical family of bus interfaces shall reference the same busDefinition in their busType subelements. They need not reference the same abstraction definitions in their abstractionType elements. See also 7.6.2 .	No	No
SCR 10.2 <i>HierFamilyBusIntfModes-Match</i>	All members of a hierarchical family of bus interfaces shall have the same interface mode (e.g., initiator , target , system). See also 7.6.2 .	No	No
SCR 10.3 <i>HierFamilyBusIntfConnReqs-Match</i>	If any member of a hierarchical family of bus interfaces has a connectionRequired element with a value of true , they all shall have this value. See also 7.6.2 .	No	No
SCR 10.4 <i>HierFamilyBusIntfSteering-Match</i>	If any member of a hierarchical family of bus interfaces has a bitSteering element with a value of 1 , they all shall have this value. See also 7.6.2 .	No	Yes

B.2.11 Hierarchy and memory maps

In a hierarchical family of bus interfaces, memory maps should be consistent.

Table B.11—Hierarchy and memory maps

Name	Rule	Single doc check	Post config
SCR 11.1 <i>hierMapSameAddresses</i>	In a hierarchical family of target or mirrored-initiator bus interfaces, all addressable bus interfaces shall define the same set of addresses to be visible. See also 7.3.1 .	No	Yes
SCR 11.2 <i>hierMapSameLocations</i>	For any member of a hierarchical family of target or mirrored-initiator bus interfaces, if an address resolves to reference a location outside the containing hierarchical family of components, that address shall reference the same location (i.e., the same address on the same bus) in every addressable member of the hierarchical family. See also 7.3.1 .	No	Yes
SCR 11.3 <i>hierMapSameProperties</i>	If any bit address (i.e., address plus bit offset) is resolved to a bit within an address block by any member of a hierarchical family of target bus interfaces, all addressable members of that family shall resolve that bit address to a bit with identical behavioral properties. See also 7.3.1 .	No	Yes
SCR 11.4 <i>hierCpuSameLocations</i>	For any member of a hierarchical family of initiator bus interfaces, if an address initiated by a cpu resolves to reference a location outside the containing hierarchical family of components, that address shall reference the same location. See 7.3.1 .	No	Yes

B.2.12 Constraints

Table B.12—Constraints

Name	Rule	Single doc check	Post config
SCR 12.1 <i>NoOutputDriveConstraint</i>	A component wire port with direction out shall not have a drive constraint. See also 6.15.16 .	Yes	No
SCR 12.2 <i>NoInputLoadConstraint</i>	A component wire port with a direction in shall not have a load constraint. See also 6.15.6.2 .	Yes	No
SCR 12.3 <i>NoOutputDriveModeConstraint</i>	An onInitiator , onTarget , or onSystem element of a wire port with direction out shall not contain a drive constraint within its modeConstraint element. See also 6.15.16 .	Yes	No
SCR 12.4 <i>NoInputLoadModeConstraint</i>	An onInitiator , onTarget , or onSystem element of a wire port with direction in shall not contain a load constraint within its modeConstraint element. See also 6.15.6.2 .	Yes	No

Table B.12—Constraints (continued)

Name	Rule	Single doc check	Post config
SCR 12.5 <i>NoOutputLoadMirroredModeConstraint</i>	An onInitiator , onTarget , or onSystem element of a wire port with direction out shall not contain a load constraint within its mirroredModeConstraint element. See also 6.15.6.2 .	Yes	No
SCR 12.6 <i>NoInputDriveMirroredModeConstraint</i>	An onInitiator , onTarget , or onSystem element of a wire port with direction in shall not contain a drive constraint with its mirroredModeConstraint element. See also 6.15.16 .	Yes	No
SCR 12.7 <i>ConstraintClockExists</i>	The clockName in a timing constraint of a component port shall be the clockName of a clockDriver or otherClockDriver element within the component unless there are no clockDriver or otherClockDriver elements present. In that case, the clockName shall be the name of another component port. See also 6.15.16.2 .	Yes	No
SCR 12.8 <i>ConstraintClockLogicalPortExists</i>	The clockName in a timing constraint of a port within an abstraction definition shall be the name of another port of the abstraction definition; that referenced port shall have an isClock subelement. See also 5.6.2 and 6.15.16.2 .	Yes	No
SCR 12.9 <i>DriverSingleBitCondition</i>	Only a scalar port or single-bit bussed port may have a clockDriver or a singleShotDriver subelement. See also 6.15.12.2 .	Yes	No

B.2.13 Design configurations

Table B.13—Design configurations

Name	Rule	Single doc check	Post config
SCR 13.1 <i>ViewConfigInstanceExists</i>	The value of an instanceName within a viewConfiguration shall match the value of the instanceName element of a componentInstance of the design document referenced by the design configuration document containing the viewConfiguration element, or design document referenced by the designInstantiation . See also 11.2.2 .	No	No
SCR 13.2 <i>ViewConfigViewExists</i>	The value of an viewName within a viewConfiguration shall match the value of the name element of a view within the component referenced by the component instance that is itself referenced by the instanceName subelement of the viewConfiguration element. See also 11.2.2 .	No	No
SCR 13.3 <i>ViewConfigsUnique</i>	No two viewConfiguration elements within a design configuration shall reference the same view. i.e., no two viewConfiguration elements may have the same instanceName . See also 11.2.2 .	Yes	Yes
SCR 13.4 <i>AbstractorInstancesUnique</i>	No two abstractor elements within a design configuration shall have the same instanceName element values. The abstractor names shall also not overlap with component instance names within the design. See also 11.3.2 .	Yes	No
SCR 13.5 <i>adhocPortRefExists</i>	A component view shall configure a componentInstantiation, designInstantiation, and designConfigurationInstantiation such that all design adHocConnection internalPortReferences exist in the configured design and all design adHocConnection externalPortReferences exist in the configured component. See also 7.6.2 and 11.5.2 .	No	No

B.2.14 Expressions

Table B.14—Expressions

Name	Rule	Single doc check	Post config
SCR 14.1 <i>complexTiedValueExpression</i>	A valued specified as an complexTiedValueExpression shall be resolved to string values "default" or "open" or to an unsigned bit vector as specified by the SystemVerilog specification, where the vector size is determined the port slice width in the ad-hoc connection. See also C.7.1 .	No	Yes
SCR 14.2 <i>unsignedLongintExpression</i>	A value specified as an unsignedLongintExpression shall be resolved to an unsigned longint as specified by the SystemVerilog specification. See also C.7.12 .	Yes	No
SCR 14.3 <i>unsignedPositiveLongintExpression</i>	A value specified as an unsignedPositiveLongintExpression shall be resolved to an unsigned longint with a value greater than 0 as specified by the SystemVerilog specification. See also C.7.14 .	Yes	No
SCR 14.4 <i>signedLongintExpression</i>	A value specified as a signedLongintExpression shall be resolved to a signed longint as specified by the SystemVerilog specification. See also C.7.4 .	Yes	No
SCR 14.5 <i>unsignedIntExpression</i>	A value specified as an unsignedIntExpression shall be resolved to an unsigned int as specified by the SystemVerilog specification. See also C.7.11 .	Yes	No
SCR 14.6 <i>unsignedPositiveIntExpression</i>	A value specified as an unsignedPositiveIntExpression shall be resolved to an unsigned int with a value greater than 0 as specified by the SystemVerilog specification. See also C.7.13 .	Yes	No
SCR 14.7 <i>realExpression</i>	A value specified as a realExpression shall resolve to a real as specified by the SystemVerilog specification. See also C.7.3 .	Yes	No
SCR 14.8 <i>stringExpression</i>	A value specified as a stringExpression shall be resolved to a string as specified by the SystemVerilog specification. See also C.7.5 .	Yes	No
SCR 14.9 <i>unsignedBitExpression</i>	A value specified as an unsignedBitExpression shall be resolved to an unsigned bit as specified by the SystemVerilog specification. See also C.7.9 .	Yes	No
SCR 14.10 <i>unsignedBitVectorExpression</i>	A value specified as an unsignedBitVectorExpression shall be resolved to an unsigned bit vector as specified by the SystemVerilog specification, where the vector size is determined by an external value (e.g., field-size for reset-value). See also C.7.10 .	Yes	No
SCR 14.11 <i>parameterExpression</i>	A parameter expression (complexBaseExpression) shall be resolved to the SystemVerilog type and sign specified for the specific parameter. See also C.7 .	Yes	No
SCR 14.12 <i>sformatfArgCount</i>	When \$sformatf() is used within an accessHandle element the invocation must have at least as many arguments beyond the format argument as the format has escapes which require an argument. See also C.1 and E.5.3.1 .	Yes	No

Table B.14—Expressions (continued)

Name	Rule	Single doc check	Post config
SCR 14.13 <i>sformatfArgType</i>	When <code>\$sformatf()</code> is used within an accessHandle element the type of an argument must match the format escape allowed arguments. See also C.1 and E.5.3.1 .	Yes	No
SCR 14.14 <i>Escape sequences</i>	The escape sequences (not functions) <code>\$ipxact_index_value</code> and <code>\$ipxact_parameter_value</code> can only be used within values of <code>displayName</code> , <code>description</code> , and <code>shortDescription</code> elements. See also E.5.5 .	Yes	No
SCR 14.15 <code>\$ipxact_index_value</code>	The function <code>\$ipxact_index_value()</code> can only be used within values of pathSegment elements. See also C.1 and E.5.4 .	Yes	No
SCR 14.16 <code>\$ipxact_field_value</code>	The function <code>\$ipxact_field_value()</code> can only be used within values of mode condition elements. See also E.5.4 .	Yes	No
SCR 14.17 <code>\$ipxact_port_value</code>	The function <code>\$ipxact_port_value()</code> can only be used within values of mode condition elements. See also E.5.4 .	Yes	No
SCR 14.18 <code>\$ipxact_mode_condition</code>	The function <code>\$ipxact_mode_condition()</code> can only be used within values of mode condition elements. See also E.5.4 .	Yes	No
SCR 14.19 <code>\$ipxact_packetfield_value</code>	The function <code>\$ipxact_packetfield_value()</code> can only be used within values of packetField width elements. See also E.5.4 .	Yes	No
SCR 14.20 <code>\$ipxact_absdefport_value</code>	The function <code>\$ipxact_absdefport_value()</code> can only be used within values of packetField width elements. See also E.5.4 .	Yes	No
SCR 14.21 <i>IndexValueIndexVarExists</i>	When using <code>\$ipxact_index_value(<indexVar>)</code> in a string expression or escape sequence, the referenced <code><indexVar></code> must exist as <code>ipxact:dim/@indexVar</code> within the object hierarchy of the containing memoryMap . See also C.7.8 and E.5.4 .	No	No
SCR 14.22 <i>ModeValueMust Exist</i>	When using <code>\$ipxact_mode_condition(<mode>)</code> in a string expression the referenced <code><mode></code> must exist as a visible mode within the containing component. See also 6.10.2 and E.5.4 .	No	No
SCR 14.23 <i>ParameterValueParameterExists</i>	When using escape sequence <code>\$ipxact_parameter_value(<parameter id>)</code> , the referenced parameter with <code>id==<parameter-id></code> must be defined within the scope of the containing top level element. See also C.21.2 .	No	No
SCR 14.24 <i>indexUniqueness</i>	The value of the <code>indexVar</code> attribute on a dim element must be unique within its scope. See also C.7.8 .	No	No
SCR 14.25 <i>qualifiedExpression</i>	The value of a qualifiedExpression in a defaultValue of a wire port shall resolve to an unsignedBitExpression , unsignedBitVectorExpression , realExpression , or realVectorExpression if the port is digital and scalar, digital and vector, analog and scalar, or analog and vector, respectively. See also 6.15.10.2 .	Yes	Yes
SCR 14.26 <i>FieldValueFieldSliceExists</i>	When using <code>\$ipxact_fieldslice_value(<fieldSliceRef>)</code> in a string of a mode condition, the referenced <code><fieldSliceRef></code> must exist as <code>ipxact:fieldSlice</code> in the encapsulating mode element. See also E.5.4 .	No	No

Table B.14—Expressions (continued)

Name	Rule	Single doc check	Post config
SCR 14.27 <i>PortValuePortSliceExists</i>	When using \$ipxact_port_value(<portSliceRef>) in a string expression of a mode condition, the referenced <portSliceRef> must exist as ipxact:portSlice in the encapsulating mode element. See also E.5.4 .	No	No
SCR 14.28 <i>PortValueAbsDefPortExists</i>	When using \$ipxact_absdefport_value(<portRef>) in a packetField the referenced <portRef> must exist as ipxact:port of the containing abstraction definition. See also E.5.4 .	No	No
SCR 14.29 <i>PacketFieldValuePacketFieldExists</i>	When using \$ipxact_packetfield_value(<packetFieldRef>) in a packetField width the referenced <packetFieldRef> must exist as ipxact:packetField within the object hierarchy of the containing ipxact:packet . See also E.5.4 .	Yes	No
SCR 14.30 <i>unresolvedStringExpressionValid</i>	Expressions of type unresolvedStringExpression shall be valid IP-XACT expressions even though they cannot be evaluated. See also C.7.6 .	Yes	No
SCR 14.31 <i>unresolvedUnsignedBitExpressionValid</i>	Expressions of type unresolvedUnsignedBitExpression shall be valid IP-XACT expressions even though they cannot be evaluated. See also C.7.7 .	Yes	No
SCR 14.32 <i>unresolvedUnsignedPositiveIntExpressionValid</i>	Expressions of type unresolvedUnsignedPositiveIntExpression shall be valid IP-XACT expressions even though they cannot be evaluated. See also C.7.8 .	Yes	No

B.2.15 Access handles

Table B.15—Access handles

Name	Rule	Single doc check	Post config
SCR 15.1 <i>accessHandleIndex</i>	The indices specified for the accessHandle should reference an index within the bounds specified by the array element when specified for a port. See also C.1.3.2 .	Yes	Yes
SCR 15.2 <i>accessHandleSlice</i>	It is not allowed to specify more than one accessHandle/slices/slice element, and it is not allowed to specify a slice/range on an addressBlock with a usage of register or reserved . See also C.1.2.2 , C.1.3.2 , and C.1.4.2 .	Yes	Yes
SCR 15.3 <i>ClearBoxElementRefExists</i>	A clearboxElementRef , which references a clearboxElement with a clearboxType of pin , shall have a pathName that is a port in the containing description. See also 6.19 .	Yes	No

Annex C

(normative)

Common elements and concepts

This annex details common elements and concepts that appear many times throughout the standard.

C.1 accessHandles

All ports and memory mapped objects can specify a list of **accessHandle** elements. Each **accessHandle** stores a portion of the HDL path for the parent object. There is a list of **accessHandles** associated with each memory mapped object to

- store a different HDL path for each view of a component (see [6.15.1.2](#)). This is specified by the **accessHandle viewRef**.
- indicate there are multiple copies of the memory mapped object within the specified view. This is indicated by two or more **accessHandle** elements that have the same **viewRef** or have no **viewRef** child elements. See [C.29](#).

The HDL path for a particular memory mapped object (e.g., a field) is distributed across the memory map hierarchy. To calculate the HDL path relative to the parent component, it is necessary to traverse from the leaf memory map object up the memory map hierarchy, concatenating the HDL path stored in the **accessHandle** associated with each memory map object.

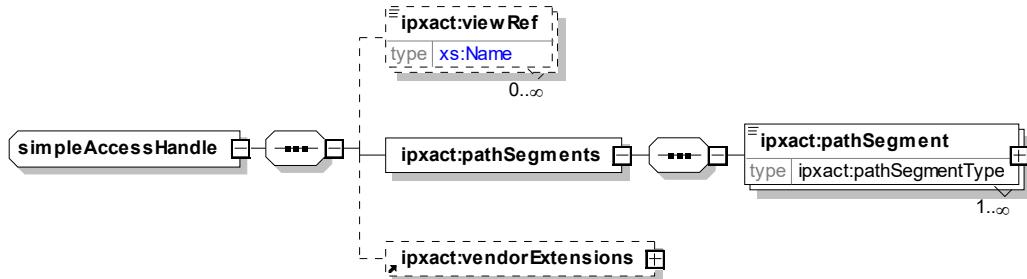
There are several types of **accessHandle** elements:

- A **simpleAccessHandle** can be used in **bank**, **registerFile**, **register**, and **alternateRegister** elements. See [C.1.1](#).
- A **slicedAccessHandle** can be used in **addressBlock** and **field** elements. See [C.1.2](#).
- A **portAccessHandle** can be used in **port** elements. See [C.1.3](#).

C.1.1 simpleAccessHandle

C.1.1.1 Schema

The following schema details the information contained in an **accessHandle** of type *simpleAccessHandle*.



C.1.1.2 Description

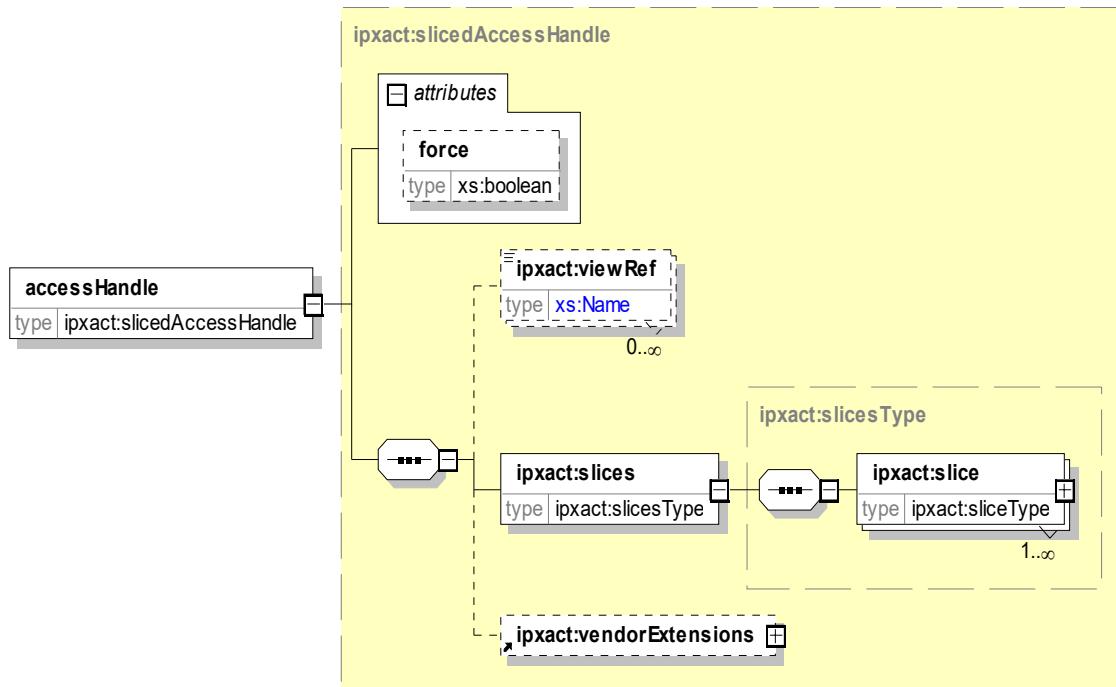
The **simpleAccessHandle** stores a portion of a HDL path for a memory mapped object. It contains the following elements:

- viewRef** (optional; type: *Name*) specifies the list of one or more views to which this **accessHandle** applies. If none is specified, the **accessHandle** is presumed to apply to all views. See [C.29](#).
- pathSegments** (mandatory) is a language-independent mechanism for referencing variables in a view. See [C.23.1](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See [C.27](#).

C.1.2 slicedAccessHandle

C.1.2.1 Schema

The following schema details the information contained in an **accessHandle** of type **slicedAccessHandle**.



C.1.2.2 Description

A **slicedAccessHandle** stores a portion of a HDL path for a memory mapped object and enables slicing in the HDL path. The **slicedAccessHandle** type contains the following attributes and elements:

- force** (optional; type: *boolean*; default: **true**) indicates if it is possible to directly write to this object via a back door access.
- viewRef** (optional; type: *Name*) specifies the list of one or more views to which this **accessHandle** applies. If none is specified, the **accessHandle** is presumed to apply to all views. See [C.29](#).
- slices** (mandatory) specifies a list of **slice** elements. See [C.1.4](#).
 Multiple **slice** elements shall not be used when the parent **accessHandle** is associated with an **addressBlock** with a type of **register** or **reserved**.

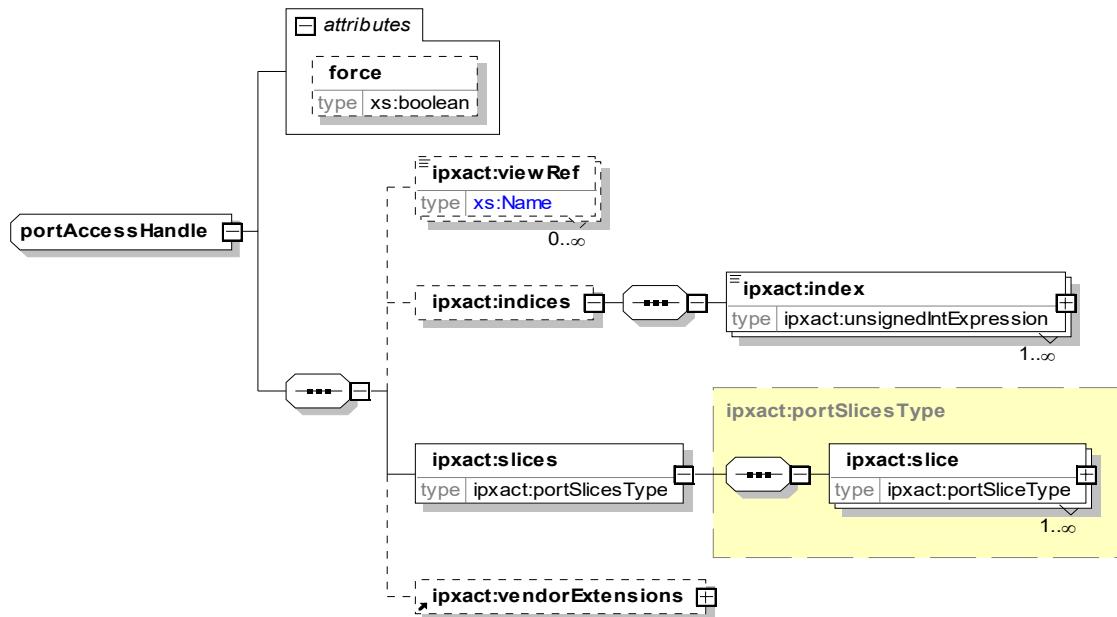
- d) **vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See [C.27](#).

See also [SCR 15.2](#).

C.1.3 portAccessHandle

C.1.3.1 Schema

The following schema details the information contained in an **accessHandle** of type **portAccessHandle**.



C.1.3.2 Description

A **portAccessHandle** describes how to access the associated IP-XACT object, e.g., a **port** in a **view** or **views**. The **portAccessHandle** type contains the following attributes and elements:

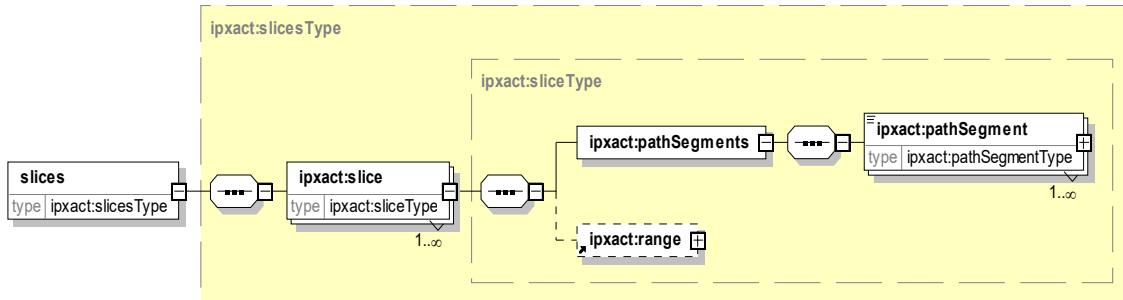
- force** (optional; type: `boolean`; default: `true`) indicates if it is possible to directly write to this object via a back door access.
- viewRef** (optional; type: `Name`) specifies the list of one or more views to which this **accessHandle** applies. If none is specified, the **accessHandle** is presumed to apply to all views. See [C.29](#).
- indices** (optional) specifies a list of **index** elements. The **indices** specify an element in the IP-XACT object to which this **accessHandle** applies. See [C.17](#).
- slices** (mandatory) specifies a list of slices. See [C.1.4](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See [C.27](#).

See also [SCR 15.1](#) and [SCR 15.2](#).

C.1.4 sliceType

C.1.4.1 Schema

The following schema details the information contained in the **slices** element.



C.1.4.2 Description

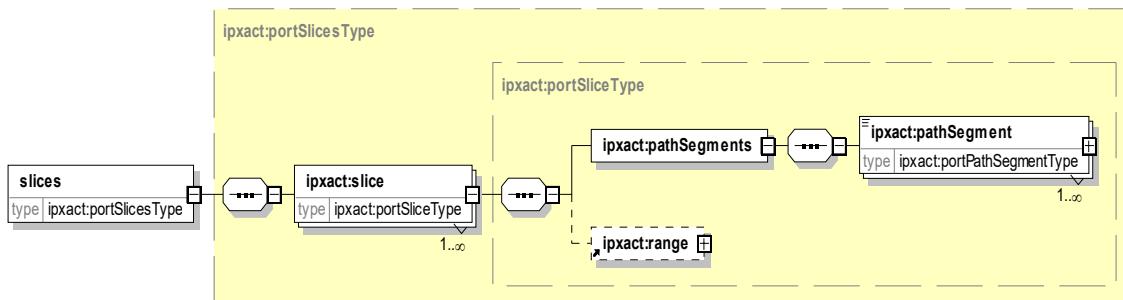
The **slices** element specifies a list of **slices**. A **slice** consists of one or more **pathSegments**. If only one **slice** is present, the IP-XACT object is represented as one variable in the view. If there are multiple **slices**, the IP-XACT object is represented by several variables in the view. In this case, the calculated path for each **slice** is concatenated in order to define the total set of bits in the IP-XACT object. Concatenation is from msb to 1sb. The **slice** element contains the following elements:

- pathSegments** (mandatory) is a language-independent mechanism for referencing variables in a view. See [C.23](#).
- range** (optional) corresponds to a range (bit select) into a variable in the RTL, i.e., the referenced view. See [C.25](#).

C.1.5 portSliceType

C.1.5.1 Schema

The following schema details the information contained in the **portSlicetypes** element.



C.1.5.2 Description

The **slices** element specifies a list of **slices**. A **slice** consists of one or more **pathSegments**. If only one **slice** is present, the IP-XACT object is represented as one variable in the view. If there are multiple **slices**, the IP-XACT object is represented by several variables in the view. In this case, the calculated path for each **slice** is

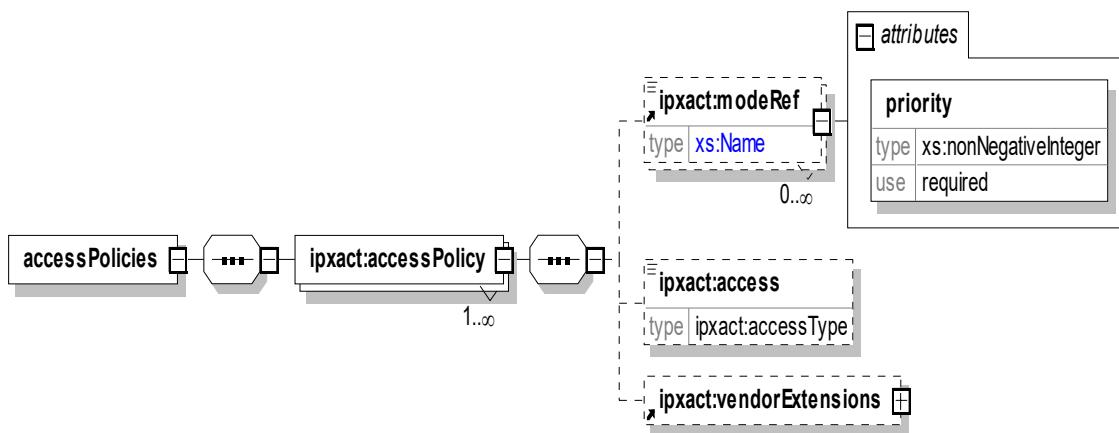
concatenated in order to define the total set of bits in the IP-XACT object. Concatenation is from msb to lsb. The **slice** element contains the following elements:

- pathSegments** (mandatory) is a language-independent mechanism for referencing variables in a view. See [C.23](#).
- range** (optional) corresponds to a range (bit select) into a variable in the RTL, i.e., the referenced view. See [C.25](#).

C.2 accessPolicies

C.2.1 Schema

The following schema details the information contained in the **accessPolicies** element.



C.2.2 Description

The **accessPolicies** element, a container for **accessPolicy**, describes the **addressBlock**, **bank**, **registerFile**, **register**, and **alternateRegister** access for different operating modes. The **accessPolicy** element contains the following elements:

- modeRef** (optional; type *Name*) identifies the operating **mode** for which the **accessPolicy** element is active by referencing a mode name in the containing description and providing a priority. An **accessPolicy** is active if it references an active mode and that has highest priority among the active modes referenced in all **accessPolicy** elements of the **accessPolicies** element. The **modeRef** element value shall be unique within the containing **accessPolicies** element. The **modeRef** element has an attribute **priority** (mandatory; type *nonNegativeInteger*) which indicates the priority of the referenced mode for this **accessPolicy**. The lower the value, the higher the priority. The priority value of a **modeRef** element shall be unique in the scope of the referenced modes inside the **accessPolicies** element. If no **modeRef** element is present, then the **accessPolicy** element applies to all modes.
- access** (optional) indicates the accessibility of the register. If an access policy references modes then the access policy element applies to the referenced modes. If an access policy does not reference a mode, then the access policy element applies as default for modes that are not referenced in other **accessPolicy** elements of the encapsulating **accessPolicies** element. If an element does not have

access policies then the access value defaults to read-write. This element can take one of the following values:

- 1) **read-write**: Both read and write transactions may have an effect on this register. Write transactions may affect the contents of the register, and read transactions return a value related to the values in the register.
 - 2) **read-only**: A read transaction to this address returns a value related to the values in the register. A write transaction to this register has undefined results.
 - 3) **write-only**: A write transaction to this address affects the contents of the register. A read transaction to this register has undefined results.
 - 4) **read-writeOnce**: Both read and write transactions may have an effect on this register. Only the first write transaction, after an event that caused the reset value of the register to be loaded, may affect the contents of the register, and read transactions return a value related to the values in the register.
 - 5) **writeOnce**: Only the first write transaction, after an event that caused the reset value of the register to be loaded, affects the contents of the register. A read transaction to this register has undefined results.
 - 6) **no-access**: Both read and write transactions to this register have undefined results.
- c) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See [C.27](#).

C.3 Access resolution

Transactions into component memory maps are filtered. To this end, component mode references in the following elements are relevant:

- **addressBlock/accessPolicies** (see [6.12.4](#) and [6.12.6](#))
- **bank/accessPolicies** (see [6.12.5](#) and [6.12.7](#))
- **register/accessPolicies** (see [6.14.3](#))
- **alternateRegister/accessPolicies** (see [6.14.5](#))
- **registerFile/accessPolicies** (see [6.14.7](#))
- **field/fieldAccessPolicies** (see [6.14.9](#))
- **accessRestrictions** (see [6.14.11](#))

Component mode conditions and modeRef priority values determine which **accessPolicy**, **fieldAccessPolicy**, and **accessRestriction** elements apply for transactions in different operation modes.

The **access** element values in **accessPolicy** elements and **fieldAccessPolicy** elements, and the **readAccessMask** en **writeAccessMask** element values in **accessRestriction** elements are applied as filters to compute the final register field bit access. The default access is **read-write** and access value **write-only**, **read-only**, and **no-access** filter out read access, write access, or both, respectively. In addition, access values **read-writeOnce** and **writeOnce** filter out multiple write access. In this way, the final register access and register field access values are computed. Finally, the **readAccessMask** en **writeAccessMask** element values filter out read and write access, respectively, for bits within register fields. [Table C.1](#) illustrates this filtering.

Table C.1—Filtering of read and write access depending on access value

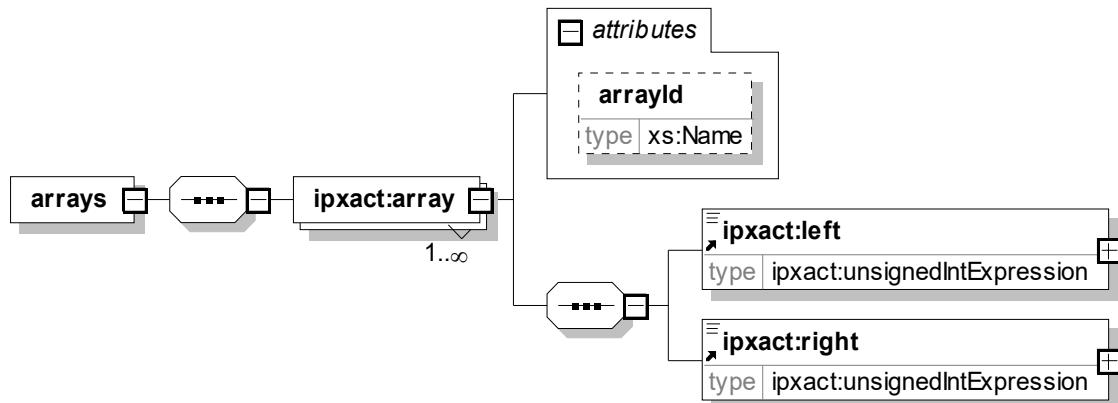
Access value	Read prevented	Write prevented
read-write	No	No
read-only	No	Yes
write-only	Yes	No
read-writeOnce	No	No, for 1st write after reset. Yes, for subsequent writes
writeOnce	Yes	No, for 1st write after reset. Yes, for subsequent writes
no-access	Yes	Yes

C.4 arrays

C.4.1 Configurable arrays with arrayId

C.4.1.1 Schema

The following schema details the information contained in the **arrays** type.



C.4.1.2 Description

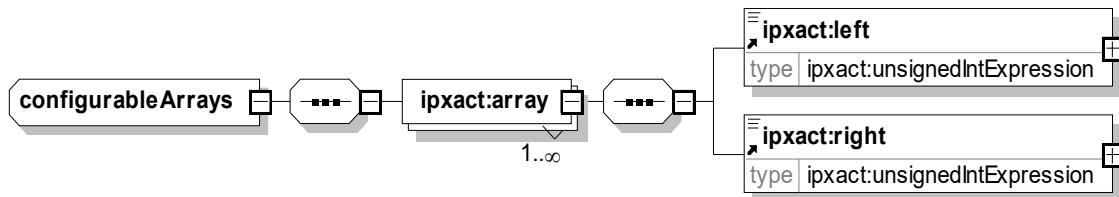
The **arrays** type specifies a set of **arrays**. An **array** element contains the following elements:

- arrayId** (optional; type: *Name*) identifies the array element for reference in the constrained attribute. See [6.15.11.2](#).
- left** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the left range for the bit slice from a parameter or port.
- right** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the right range for the bit slice from a parameter or port.

C.4.2 Configurable arrays without arrayId

C.4.2.1 Schema

The following schema details the information contained in the **configurableArrays** type.



C.4.2.2 Description

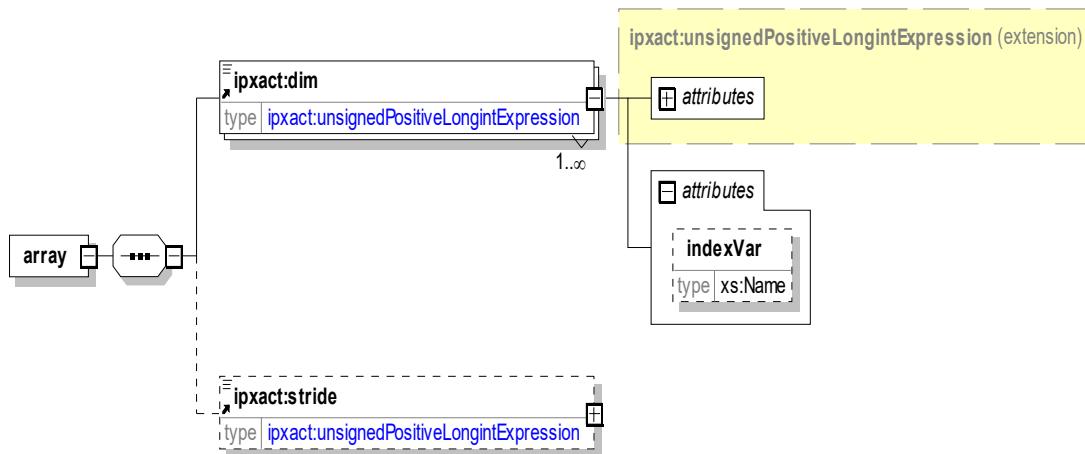
The **configurableArrays** type specifies a set of **arrays**. An **array** element contains the following elements:

- left** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the left range for the bit slice from a parameter or port.
- right** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the right range for the bit slice from a parameter or port.

C.4.3 Memory arrays with stride

C.4.3.1 Schema

The following schema details the information contained in the **array** element.



C.4.3.2 Description

The **array** element contains the following elements:

- dim** (mandatory; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) assigns an unbounded dimension to its encapsulating element (**addressBlock**, **registerFile**, or **register**), so it is repeated as many times as the value of the **dim** elements. For multi-dimensional elements, the memory layout is presumed to follow the IEEE Std 1666 [\[B4\]](#) (SystemC) language rules.

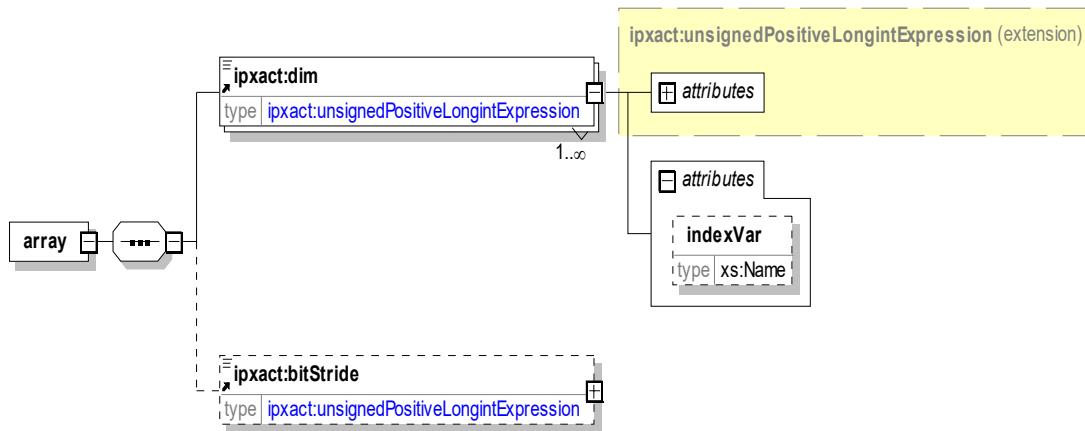
The attribute **indexVar** (optional; type *Name*) provides a name for the dimension. It can be used as argument in \$ipxact_index_value function calls or escape sequences.

- b) **stride** (optional; type *unsignedPositiveLongintExpression*, see [C.7.14](#)) describes the distance between two consecutive array elements in **addressUnitBits** from the containing element. If not present, the stride value is assumed to be equal to the **addressBlock/range**, **registerFile/range**, or the least integer k such that $k \times addressUnitBits$ is at least the **register/size** value, depending on whether the containing element is an **addressBlock**, **registerFile**, or **register**, respectively.

C.4.4 Field arrays with bit stride

C.4.4.1 Schema

The following schema details the information contained in the **array** element.



C.4.4.2 Description

The **array** element contains the following elements:

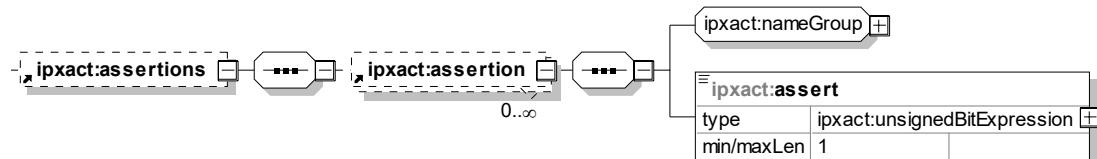
- a) **dim** (mandatory; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) assigns an unbounded dimension to the encapsulating field, so it is repeated as many times as the value of the dim elements. For multi-dimensional fields, the memory layout is presumed to follow the IEEE Std 1666 [B4] (SystemC) language rules.

The attribute **indexVar** (optional; type: *Name*) provides a name for the dimension. It can be used as argument in \$ipxact_index_value function calls or escape sequences.
- b) **bitStride** (optional; type: *unsignedPositiveLongintExpression*, see [C.7.14](#)) describes the distance between two consecutive array elements in bits. If not present, the stride value is assumed to be equal to the bit-Width value.

C.5 assertions

C.5.1 Schema

The following schema details the information contained in the **assertions** element.



C.5.2 Description

The **assertions** element contains an unbounded list of **assertion** elements. An **assertion** describes the allowed parameter values as an assertion expression. If this expression evaluates to 0, the **name**, **displayName**, and/or **description** (of the **nameGroup**) can be used to relay that the assertion failed. The **assertion** element definition contains the following elements:

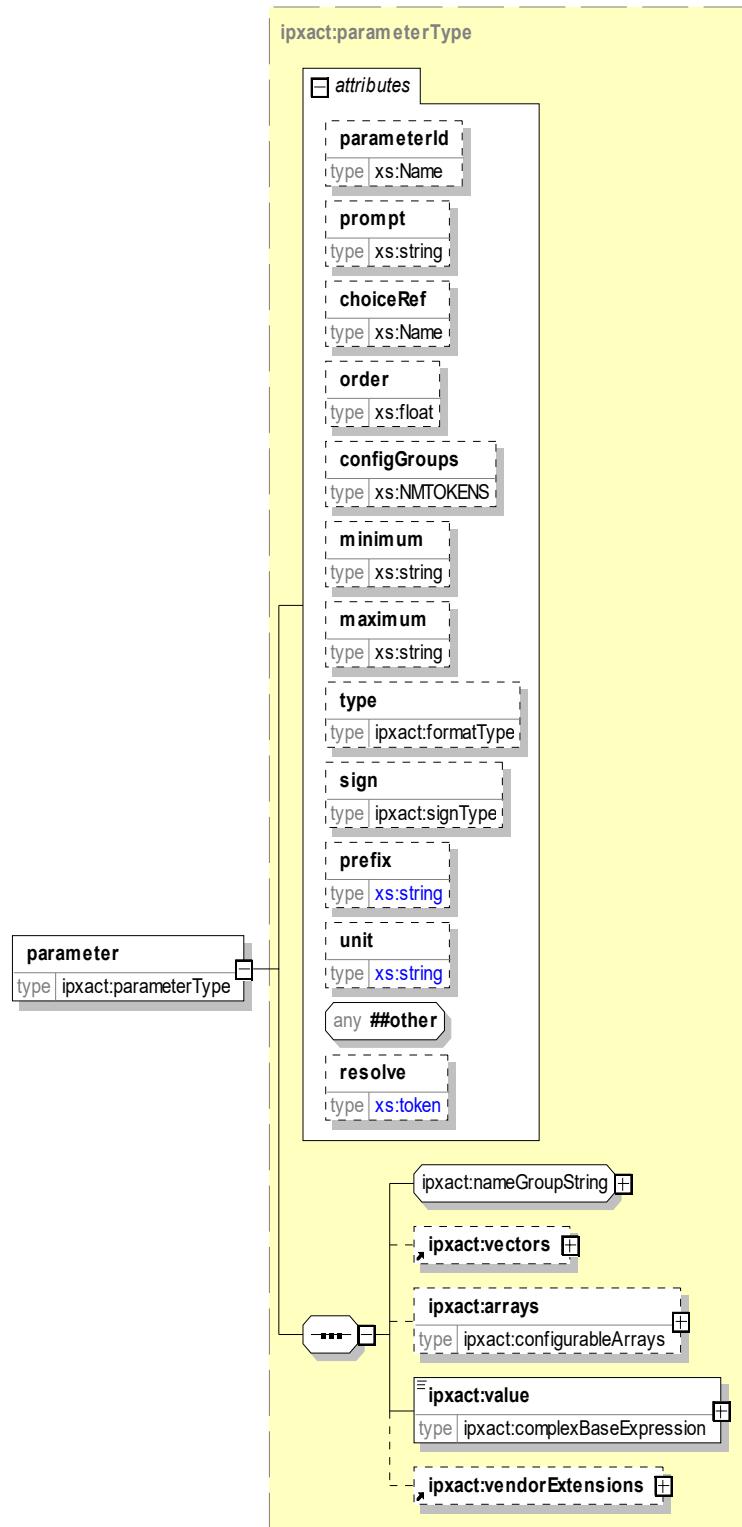
- nameGroup** is defined in [C.19](#).
- assert** (mandatory; type: *unsignedBitExpression*, see [C.7.9](#)) contains an expression that is expected to evaluate to **1**.

See also [SCR 5.26](#).

C.6 attributes

C.6.1 Schema

The following schema details the information contained in the *attributes* for various elements.



C.6.2 Description

The **parameter** attributes specify the parameter behavior via the following elements:

- a) The **parameterId** attribute (optional; type: *Name*) assigns a unique identifier to the containing parameter for reference throughout the containing description. **parameterId** is required when the element has a **resolve** type equal to **user** or **generated**, or is referenced from an expression. This **parameterId** can be referenced in two ways: by the **configurableElementValue** element (see [C.9.2](#)) in the referencing description or in an expression.
- b) The **prompt** attribute (optional; type: *string*) defines a prompt string that a DE can use if the **resolve** attribute is equal to **user**.
- c) The **choiceRef** attribute (optional; type: *Name*) indicates the value of the containing element is defined in the referenced **choice** element.
- d) The **order** attribute (optional; type: *float*) indicates how elements are presented, when **resolve** equals **user**. The elements are presented in ascending order.
- e) The **configGroups** attribute (optional; type: *NMTOKENS*) indicates a name to group elements together; elements with matching values for this attribute are contained in the same group. There is no semantic meaning to this attribute.
- f) The **minimum** attribute (optional; type: *string*) indicates the lower bound for the **value** of the containing element. This check is valid only for a format of **byte**, **shortint**, **int**, **longint**, **real**, or **shortreal**. The **type** attribute shall specify the type of the **minimum** attribute.
- g) The **maximum** attribute (optional; type: *string*) indicates the lower bound for the **value** of the containing element. This check is valid only for a format of **byte**, **shortint**, **int**, **longint**, **real**, or **shortreal**. The **type** attribute shall specify the type of the **maximum** attribute.
- h) The **type** attribute (optional; default: *string*) is the type to which the parameter value resolves. The value shall be one of the following:
 - 1) **bit** indicates the value shall resolve to a SystemVerilog **bit**, which by default is resolved to a 1-bit value, unless a vector size has been specified.
 - 2) **byte** indicates the value shall resolve to a SystemVerilog **byte**, which is resolved to an 8-bit integer value.
 - 3) **shortint** indicates the value shall resolve to a SystemVerilog **shortint**, which is resolved to a 16-bit integer value.
 - 4) **int** indicates the value shall resolve to a SystemVerilog **int**, which is resolved to a 32-bit integer value.
 - 5) **longint** indicates the value shall resolve to a SystemVerilog **longint**, which is resolved to a 64-bit integer value.
 - 6) **shortreal** indicates the value shall resolve to a SystemVerilog **shortreal**, which is resolved to a 32-bit floating point value.
 - 7) **real** indicates the value shall resolve to a SystemVerilog **real**, which is resolved to a 64-bit floating point value.
 - 8) **string** indicates the value shall resolve to a SystemVerilog **string**.
- i) The **sign** attribute (optional) indicates the signed-ness of the parameter value. This property influences only the following types: **bit**, **byte**, **shortint**, **int**, and **longint**:
where the types **byte**, **shortint**, **int**, and **longint** are signed by default and the type **bit** is unsigned by default.
The **sign** property does not influence **real**, **shortreal**, **bit**, and **string** types; effectively, **sign** can be ignored for these types.
- j) The **prefix** attribute (optional; type: *string*) specifies the prefix that precedes the **unit** of a value. The **prefix** is for informational purposes only and is not applied to the value (e.g., when evaluating expressions). Its possible values are **hecto**, **kilo**, **mega**, **giga**, **tera**, **peta**, **exa**, **zetta**, **yotta**, **deci**,

centi, milli, micro, nano, pico, femto, atto, zepto, and yocto. These represent the values as defined by the *International System of Units* [B6].

- k) The **unit** attribute (optional; type: *string*) specifies the unit of a value. The **unit** is for informational purposes only and is not applied to the value (e.g., when evaluating expressions). Its possible values are **second, ampere, kelvin, hertz, joule, watt, coulomb, volt, farad, ohm, siemens, henry, and Celsius**. These represent some of the base units and derived units as defined by the *International System of Units* [B6].
- l) The **any ##other** attribute (optional) indicates any additional attributes in other name spaces are allowed in the containing element. These additional attributes are called *vendor attributes*.
- m) The **resolve** attribute (optional; type: *token*; default: **immediate**) defines how the value for the containing element is configured. The value shall be one from the enumerated list of **immediate, user, or generated**.
 - 1) **immediate** indicates the value shall be specified in the containing element.
 - 2) **user** indicates the value shall be specified by user input and the new value stored in a referencing description under the **configurableElementValue** element (see [C.9.2](#)).
 - 3) **generated** indicates the value shall be set by a generator and the new value stored in a referencing description under the **configurableElementValue** element (see [C.9.2](#)).

See also [SCR 5.1](#).

C.7 complexBaseExpression

The **complexBaseExpression** is used as the base type for the more specific expression types, but can also be used directly by parameters. When used directly on a parameter, the value of the element needs to be specified as an expression that can resolve to the type and signed-ness specified for the containing parameter. When the value does not represent the type specified, the parser should attempt to cast the value to this type. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value; the type and signed-ness of the minimum and maximum value corresponds to the type of the parameter and signed-ness. An error should be reported when the value is specified outside the minimum and maximum range.

complexBaseExpression can be any of the following subelements:

- **complexTiedValueExpression** (see [C.7.1](#))
- **qualifiedExpression** (see [C.7.2](#))
- **realExpression** (see [C.7.3](#))
- **signedLongintExpression** (see [C.7.4](#))
- **stringExpression** (see [C.7.5](#))
- **unresolvedStringExpression** (see [C.7.6](#))
- **unresolvedUnsignedBitExpression** (see [C.7.7](#))
- **unresolvedUnsignedPositiveIntExpression** (see [C.7.8](#))
- **unsignedBitExpression** (see [C.7.9](#))
- **unsignedBitVectorExpression** (see [C.7.10](#))
- **unsignedIntExpression** (see [C.7.11](#))
- **unsignedLongintExpression** (see [C.7.12](#))
- **unsignedPositiveIntExpression** (see [C.7.13](#))
- **unsignedPositiveLongintExpression** (see [C.7.14](#))

See also [SCR 5.13](#) and [SCR 14.11](#).

C.7.1 complexTiedValueExpression

C.7.1.1 Schema

The following schema details the information contained in the *complexTiedValueExpression* element.



C.7.1.2 Description

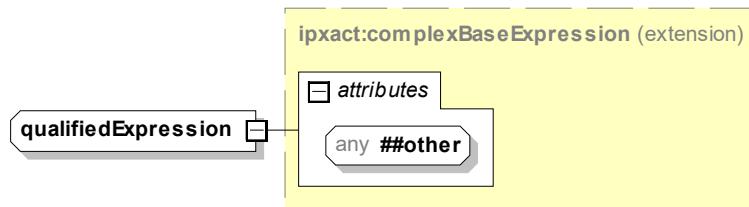
The value of the *complexTiedValueExpression* element

See also [SCR 14.1](#).

C.7.2 qualifiedExpression

C.7.2.1 Schema

The following schema details the information contained in the *qualifiedExpression* element.



C.7.2.2 Description

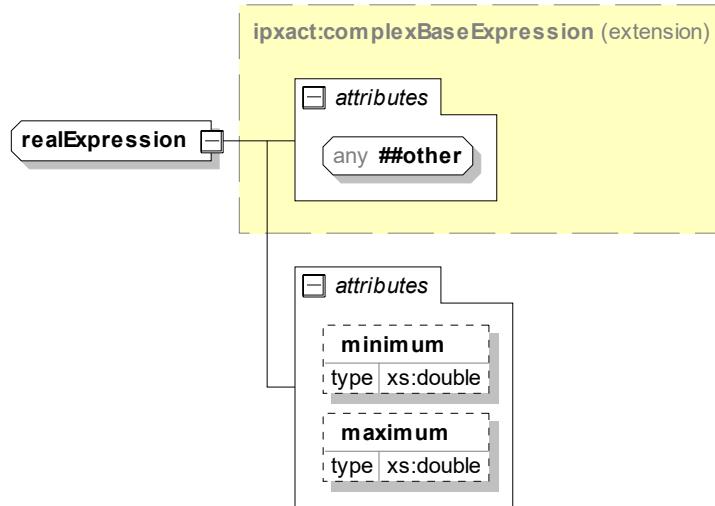
The value of the *qualifiedExpression* element needs to be specified as an expression that can resolve to an *unsignedBitExpression*, *unsignedBitVectorExpression*, *realExpression*, or *realVectorExpression*.

The type it needs to resolve to depends on the properties of the encapsulating port. A port is called digital if its **signalTypeDef** has the value digital, otherwise it is called analog. A port is a vector if it has a vector element, otherwise it is a scalar. The value of a *qualifiedExpression* in a defaultValue of a wire port shall resolve to an *unsignedBitExpression*, *unsignedBitVectorExpression*, *realExpression*, or *realVectorExpression* if the port is digital and scalar, digital and vector, analog and scalar, or analog and vector, respectively. See also [SCR 14.25](#).

C.7.3 realExpression

C.7.3.1 Schema

The following schema details the information contained in the *realExpression* element.



C.7.3.2 Description

The value of the *realExpression* element needs to be specified as an expression that can resolve to a 64-bit *real* value (as specified by IEEE Std 1800). When the value does not represent a *real*, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

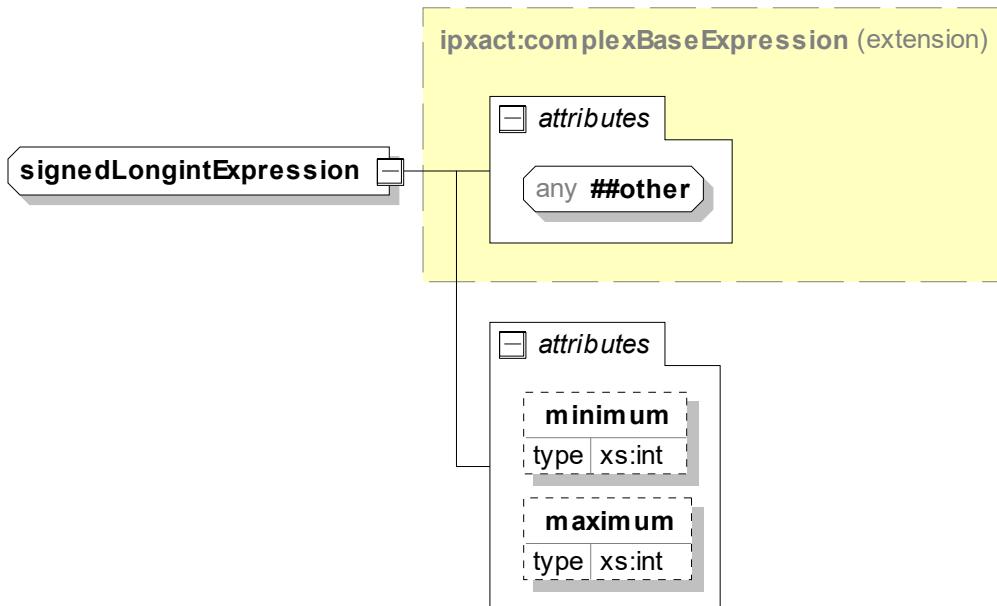
- a) **minimum** (optional) indicates the lower bound for the value of the containing element.
- b) **maximum** (optional) indicates the upper bound for the value of the containing element.

See also [SCR 14.7](#).

C.7.4 signedLongintExpression

C.7.4.1 Schema

The following schema details the information contained in the *signedLongintExpression* element.



C.7.4.2 Description

The value of the *signedLongintExpression* element needs to be specified as an expression that can resolve to a 64-bit signed longint value (as specified by IEEE Std 1800). When the value does not represent a signed longint, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

- a) **minimum** (optional) indicates the lower bound for the value of the containing element.
- b) **maximum** (optional) indicates the upper bound for the value of the containing element.

See also [SCR 14.4](#).

C.7.5 stringExpression

C.7.5.1 Schema

The following schema details the information contained in the **stringExpression** element.



C.7.5.2 Description

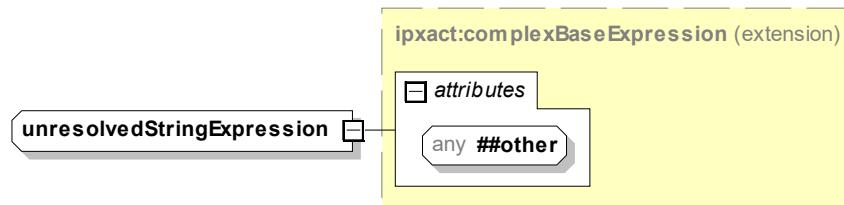
The value of the **stringExpression** element needs to be specified as an expression that can resolve to a string value (as specified by IEEE Std 1800). An error should be reported when the value does not represent a string.

See also [SCR 14.8](#).

C.7.6 unresolvedStringExpression

C.7.6.1 Schema

The following schema details the information contained in the **unresolvedStringExpression** element.



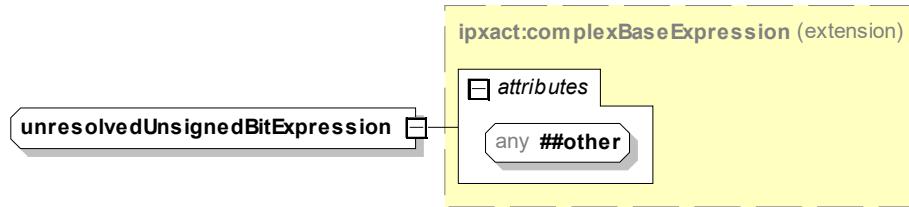
C.7.6.2 Description

The value of the **unresolvedStringExpression** element needs to be specified as an expression that can resolve to a string value (as specified by IEEE Std 1800) although (terms in) the expression may need to be resolved outside IP-XACT. An error should be reported when the value does not represent a string.

C.7.7 unresolvedUnsignedBitExpression

C.7.7.1 Schema

The following schema details the information contained in the ***unresolvedUnsignedBitExpression*** element.



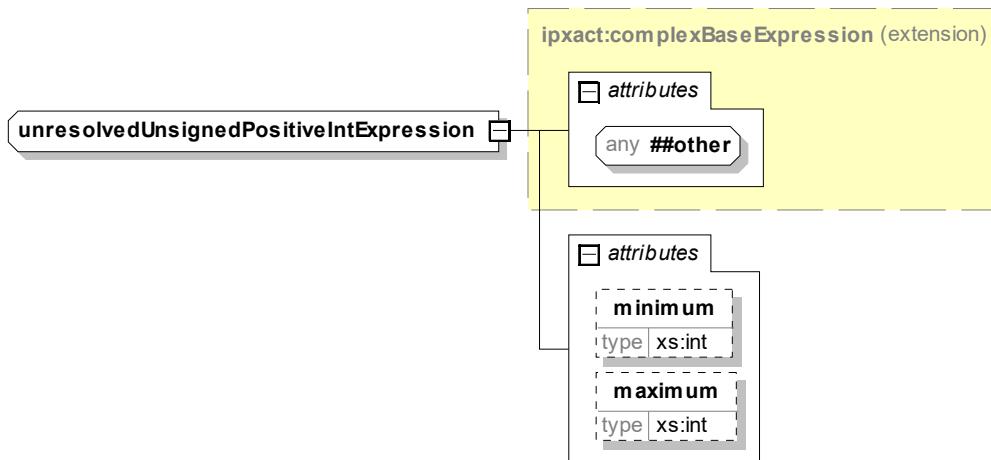
C.7.7.2 Description

The value of the ***unresolvedUnsignedBitExpression*** element needs to be specified as an expression that can resolve to a 1-bit value (1 or 0) (as specified by IEEE Std 1800) although (terms in) the expression may need to be resolved outside IP-XACT. An error should be reported when the value does not represent a string.

C.7.8 unresolvedUnsignedPositiveIntExpression

C.7.8.1 Schema

The following schema details the information contained in the ***unresolvedUnsignedPositiveIntExpression*** element.



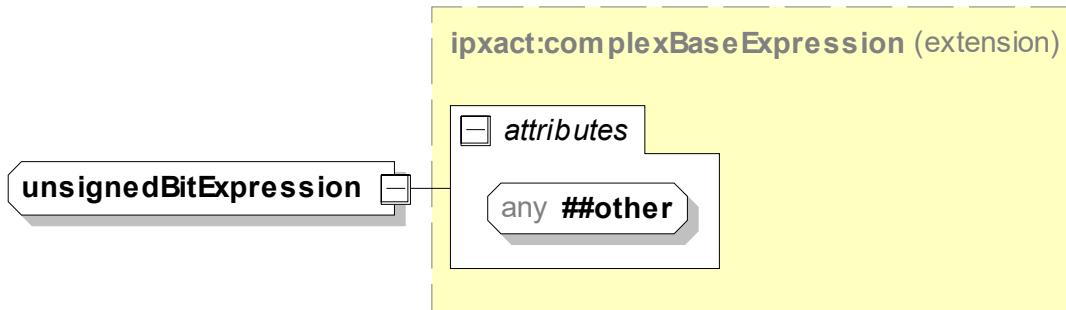
C.7.8.2 Description

The value of the ***unresolvedUnsignedPositiveIntExpression*** element needs to be specified as an expression that can resolve to a 32-bit unsigned positive int value (as specified by IEEE Std 1800) although (terms in) the expression may need to be resolved outside IP-XACT. An error should be reported when the value does not represent a string.

C.7.9 **unsignedBitExpression**

C.7.9.1 Schema

The following schema details the information contained in the **unsignedBitExpression** element.



C.7.9.2 Description

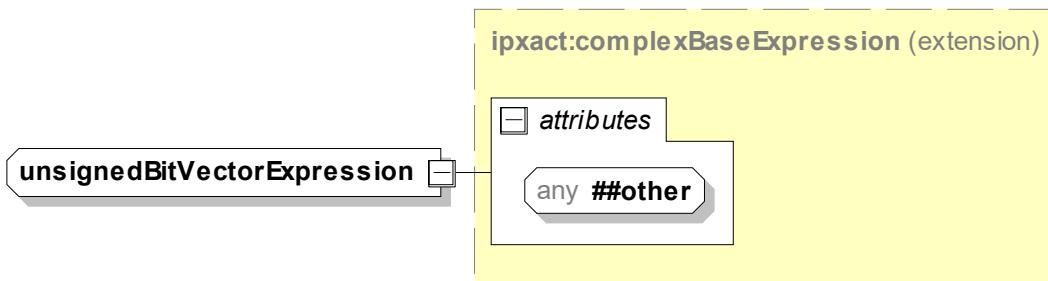
The value of the **unsignedBitExpression** element needs to be specified as an expression that can resolve to a 1-bit value (1 or 0) (as specified by IEEE Std 1800). When the value does not represent a `bit`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

See also [SCR 14.9](#).

C.7.10 **unsignedBitVectorExpression**

C.7.10.1 Schema

The following schema details the information contained in the **unsignedBitVectorExpression** element.



C.7.10.2 Description

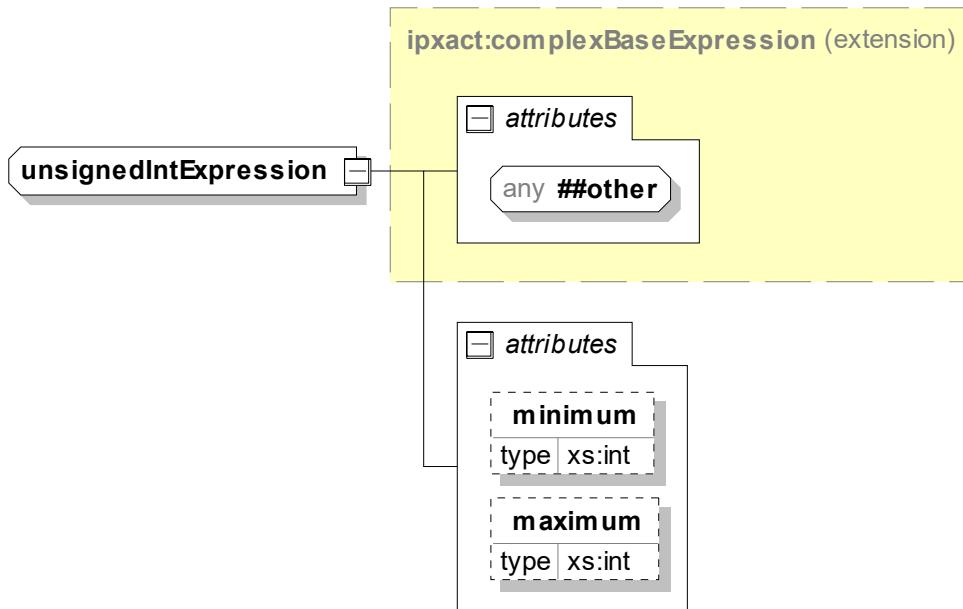
The value of the **unsignedBitVectorExpression** element needs to be specified as an expression that can resolve to a vector of `bit` (as specified by IEEE Std 1800). The length of the `bit vector` is based on the width of the containing object, i.e., the port or field. When the value does not represent a `bit vector`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

See also [SCR 14.10](#).

C.7.11 *unsignedIntExpression*

C.7.11.1 Schema

The following schema details the information contained in the *unsignedIntExpression* element.



C.7.11.2 Description

The value of the *unsignedIntExpression* element needs to be specified as an expression that can resolve to a 32-bit unsigned int value (as specified by IEEE Std 1800). When the value does not represent an unsigned int, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

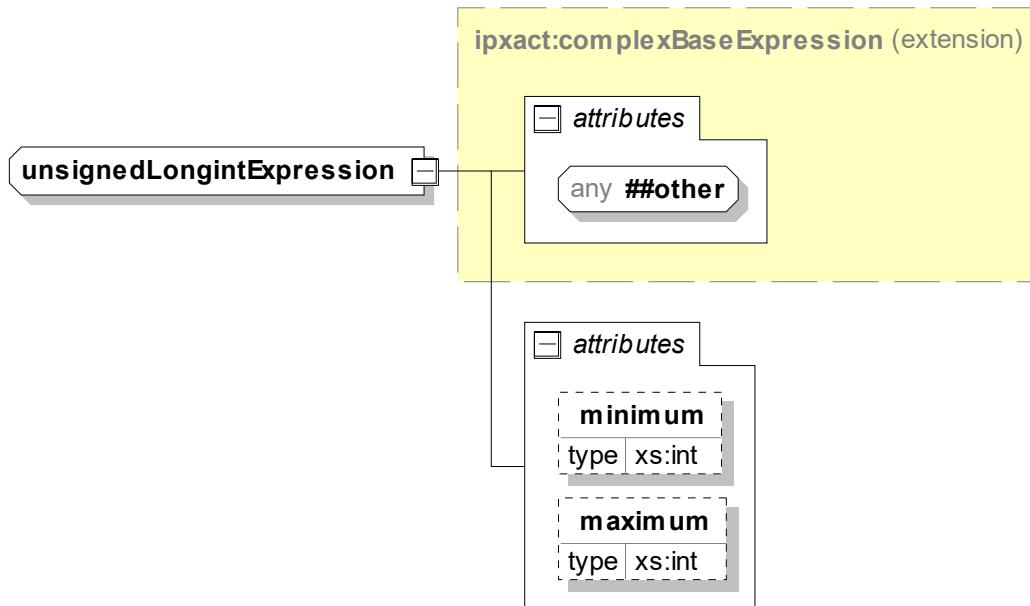
- a) **minimum** (optional) indicates the lower bound for the value of the containing element.
- b) **maximum** (optional) indicates the upper bound for the value of the containing element.

See also [SCR 14.5](#).

C.7.12 *unsignedLongintExpression*

C.7.12.1 Schema

The following schema details the information contained in the *unsignedLongintExpression* element.



C.7.12.2 Description

The value of the *unsignedLongintExpression* element needs to be specified as an expression that can resolve to a 64-bit unsigned longint value (as specified by IEEE Std 1800). When the value does not represent an unsigned longint, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

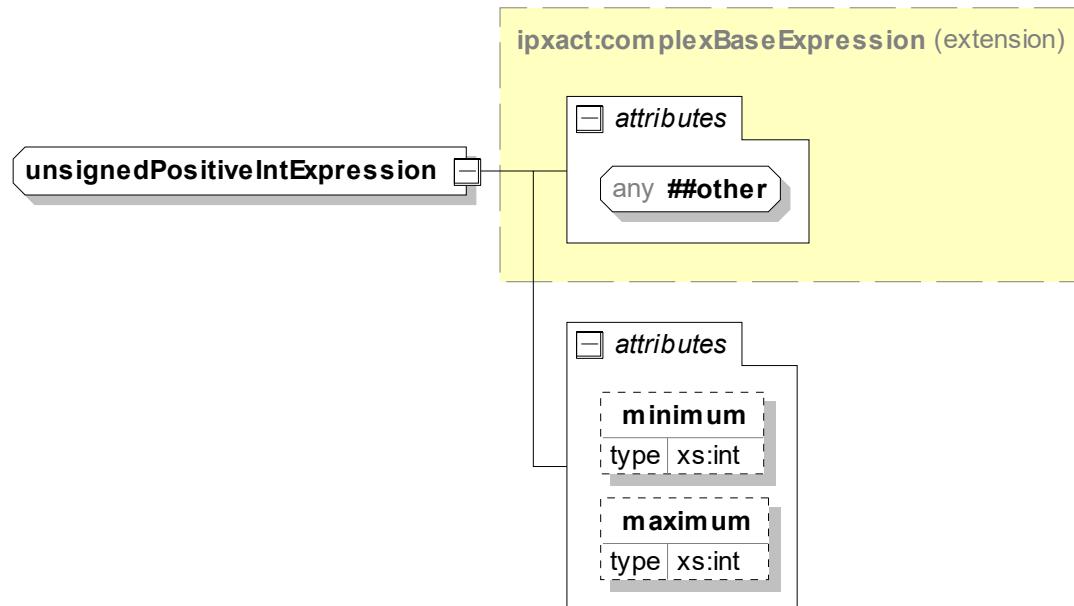
- a) **minimum** (optional) indicates the lower bound for the value of the containing element.
- b) **maximum** (optional) indicates the upper bound for the value of the containing element.

See also [SCR 14.2](#).

C.7.13 *unsignedPositiveIntExpression*

C.7.13.1 Schema

The following schema details the information contained in the *unsignedPositiveIntExpression* element.



C.7.13.2 Description

The value of the *unsignedPositiveIntExpression* element needs to be specified as an expression that can resolve to a 32-bit unsigned int value (as specified by IEEE Std 1800). When the value does not represent an unsigned int, the parser should attempt to cast the value. An error should be reported when the value cannot be cast or it is not a positive number.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

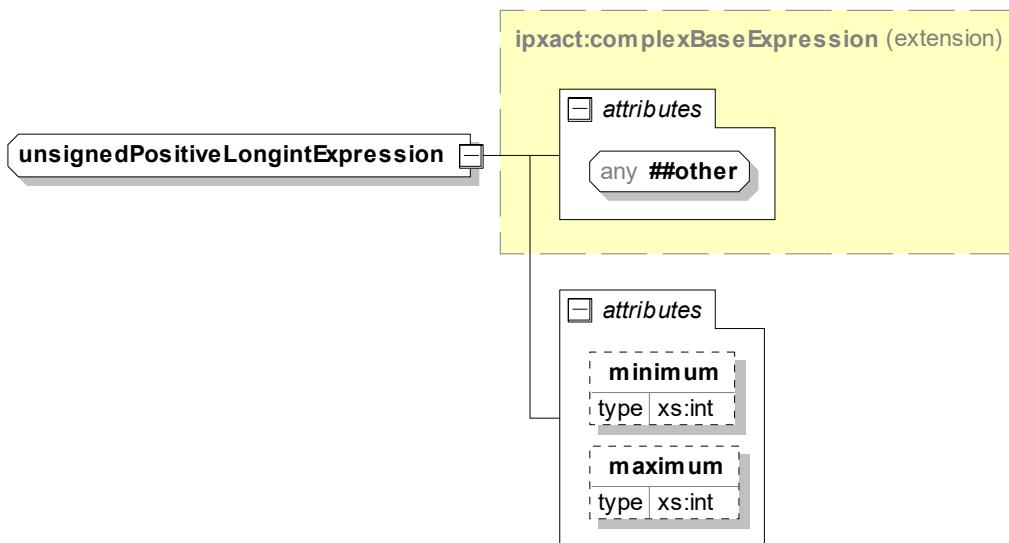
- a) **minimum** (optional) indicates the lower bound for the value of the containing element.
- b) **maximum** (optional) indicates the upper bound for the value of the containing element.

See also [SCR 14.6](#).

C.7.14 *unsignedPositiveLongintExpression*

C.7.14.1 Schema

The following schema details the information contained in the *unsignedPositiveLongintExpression* element.



C.7.14.2 Description

The value of the *unsignedPositiveLongintExpression* element needs to be specified as an expression that can resolve to a 64-bit unsigned longint value (as specified by IEEE Std 1800). When the value does not represent an unsigned longint, the parser should attempt to cast the value. An error should be reported when the value cannot be cast or it is not a positive number.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

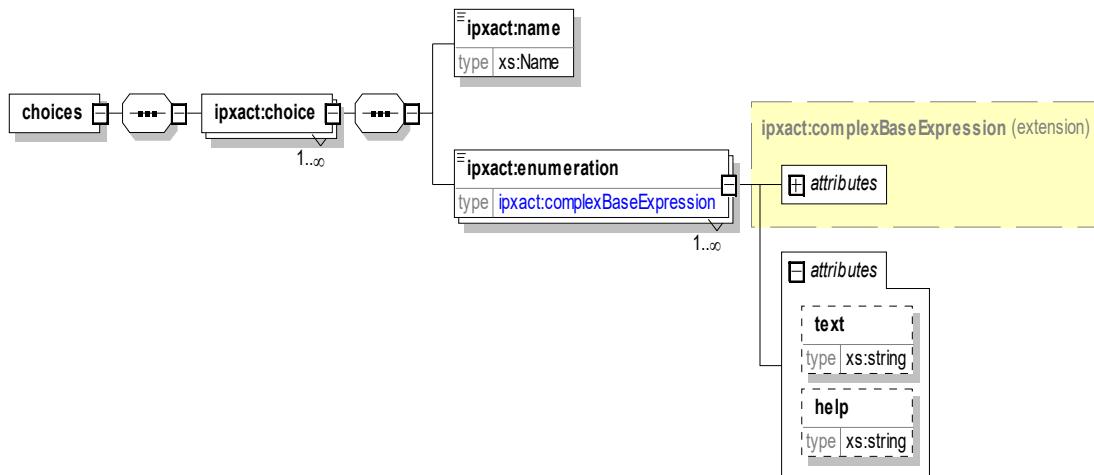
- a) **minimum** (optional) indicates the lower bound for the value of the containing element.
- b) **maximum** (optional) indicates the upper bound for the value of the containing element.

See also [SCR 14.3](#).

C.8 choices

C.8.1 Schema

The following schema details the information contained in the **choices** element, which may appear as an element inside.



C.8.1.1 Description

The **choices** element contains an unbounded list of **choice** elements. Each **choice** element is a list of items used by a **moduleParameter** element, **parameter** element, or any other configurable element with a **choiceRef** attribute. These elements indicate they are using a **choice** element by setting the attribute **choiceRef**. This **choiceRef** attribute shall reference a valid **choice/name** element in the containing description.

The **choice** definition contains the following elements:

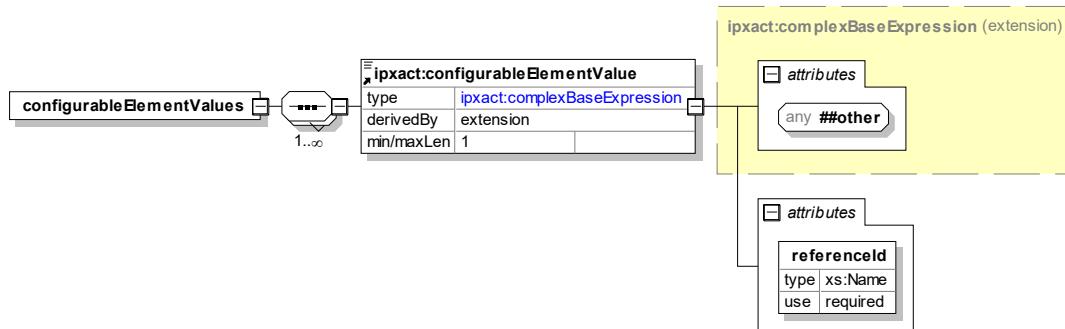
- name** (mandatory; type: *Name*) specifies the name of this list and is used by other elements for reference. The **name** elements shall be unique within the containing **choices** element.
- enumeration** (mandatory; type: *complexBaseExpression*, see [C.7](#)) is an unbounded list of elements, where each holds a possible value that the referencing element may contain.
 - text** (optional; type: *string*) attribute causes optional text to be displayed when choosing the **choice** value. The resulting value stored in the configurable element corresponds to the enumeration value for the choice. If the **text** attribute is not present, the **enumeration** value may be displayed.
 - help** (optional; type: *string*) attribute gives any additional information about this enumeration element.

See also [SCR 5.7](#).

C.9 configurableElementValues

C.9.1 Schema

The following schema details the information contained in the *configurableElementValues* element.



C.9.2 Description

The **configurableElementValues** element specifies the configuration for a specific component instance, external type definition, bus type, abstraction type, design instantiation, design configuration instantiation, or generator chain configuration by providing the value of a specific parameter. The **configurableElementValues** is an unbounded list of **configurableElementValue** elements.

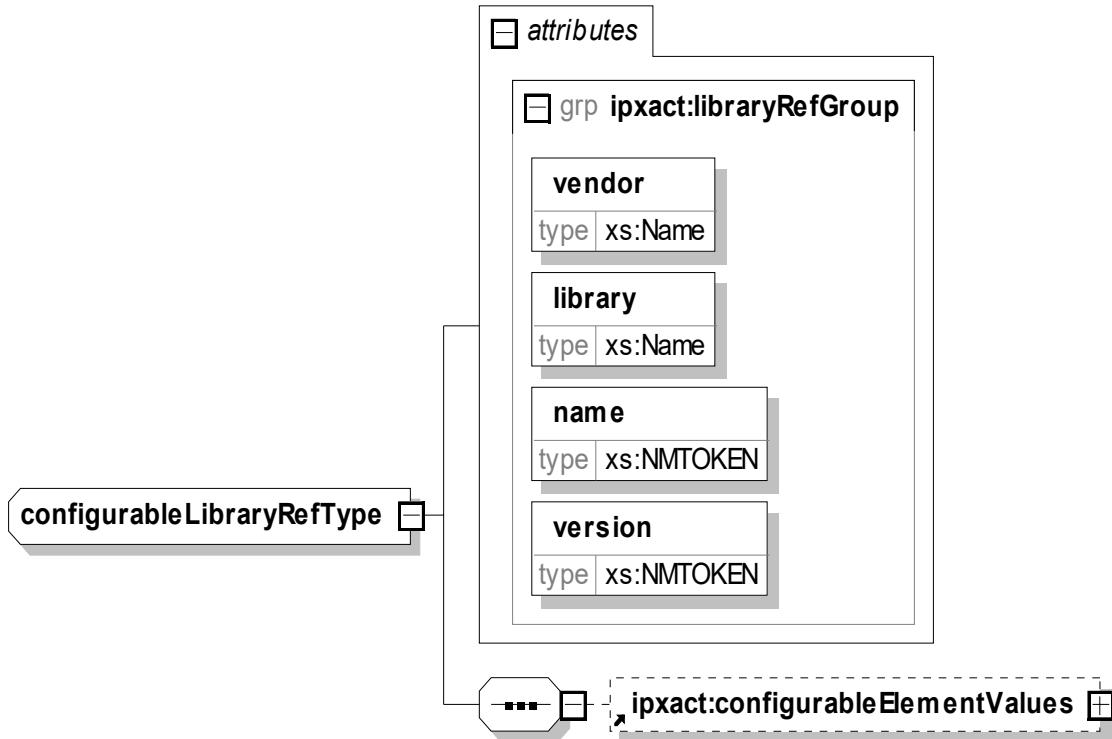
- a) **configurableElementValue** (mandatory; type: *complexBaseExtension*, see [C.7](#)) is an unbounded list that specifies the value to apply to a configurable element; in this instance, it is pointed to by the **referenceId** attribute.
- b) The contained **referenceId** (mandatory; type: *Name*) attribute is a reference to the **parameterId** attribute of an element in the component instance.

See also [SCR 5.4](#), [SCR 5.5](#), [SCR 5.6](#), [SCR 5.7](#), [SCR 5.8](#), [SCR 5.9](#), [SCR 5.10](#), and [SCR 5.11](#).

C.10 configurableLibraryRefType

C.10.1 Schema

The following schema details the information contained in the *configurableLibraryRefType* type.



C.10.2 Description

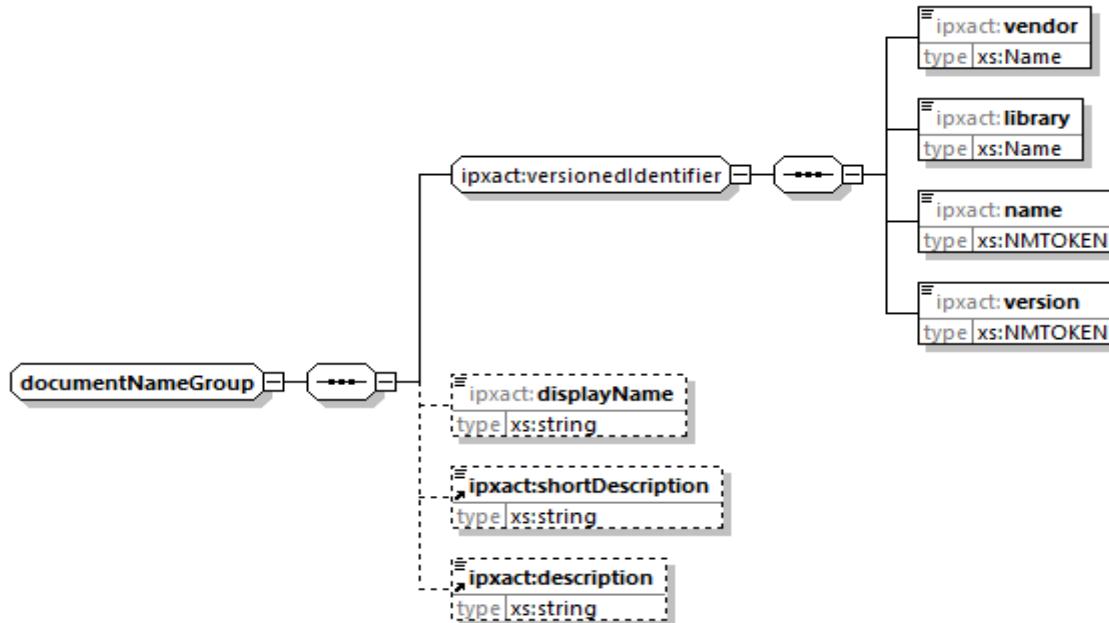
The *configurableLibraryRefType* type defines a set of four attributes that reference another IP-XACT description through the unique VLVN identifier and any configuration information.

- The **vendor** attribute (mandatory; type: *Name*) identifies the owner of the referenced description.
- The **library** attribute (mandatory; type: *Name*) identifies the library of the referenced description.
- The **name** attribute (mandatory; type: *NMTOKEN*) identifies the name of the referenced description.
- The **version** attribute (mandatory; type: *NMTOKEN*) identifies the version of the referenced description.
- configurableElementValues** (optional) specifies the configuration for a specific component instance, external type definition, bus type, abstraction type, design instantiation, design configuration instantiation, or generator chain configuration by providing the value of a specific parameter. See [C.9](#).

C.11 documentName group

C.11.1 Schema

The following schema details the information contained in the *documentName* group.



C.11.2 Description

The *documentName* group defines a unique reference of or from an IP-XACT description. Only one object with a given VLVN may be present in a DE at any given time. The timing and way to change the VLVN of an object is completely up to the user or developer. The *versionedIdentifier* group definition contains the following subelements:

- vendor** (mandatory; type: *Name*) identifies the owner of this description. The format of the **vendor** element is the company Internet domain name in left-to-right order (e.g., accellera.org not org.acellera).
- library** (mandatory; type: *Name*) identifies the library of this description. This allows a vendor to group descriptions.
- name** (mandatory; type: *NMTOKEN*) identifies the name of this description within a library.
- version** (mandatory; type: *NMTOKEN*) identifies the version of this description. This allows a vendor to provide many descriptions that all have the same name, but are still uniquely identified. The **version** may appear as an alphanumeric string and contain a set of substrings, with non-alphanumeric delimiters in-between. Each IP supplier shall have their own cataloging system for setting version numbers.
- displayName** (optional; type: *string*) allows a short descriptive text to be associated with the port. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- shortDescription** (optional; type: *string*) is a more compact textual description of the containing element compared to **description**.
- description** (optional; type: *string*) allows a textual description of the containing element. See [C.19.2](#).

See also [SCR 1.1](#) and [SCR 1.2](#).

C.12 Endianness

There are (only) two legal values (**big** and **little**) to specify the **endianness**.

- *Big endian (big)* means the most significant byte of any multi-byte data field is stored at the lowest memory address, which is also the address of the larger field.

There are at least two ways for big-endianness to manifest itself: byte-invariant and word-invariant (also known as *middle-endian*). The difference is that if data is stored as *word-invariant*, the data is stored differently for transfers larger than a byte, for example,

Byte invariant: A word access to address $0x0$ is on $D[31:0]$. The MSB is $D[7:0]$, the LSB is $D[31:24]$.

Word invariant: A word access to address $0x0$ is on $D[31:0]$. The MSB is $D[31:24]$, the LSB byte is $D[7:0]$.

In IP-XACT, the interpretation of big-endian is the byte-invariant style.

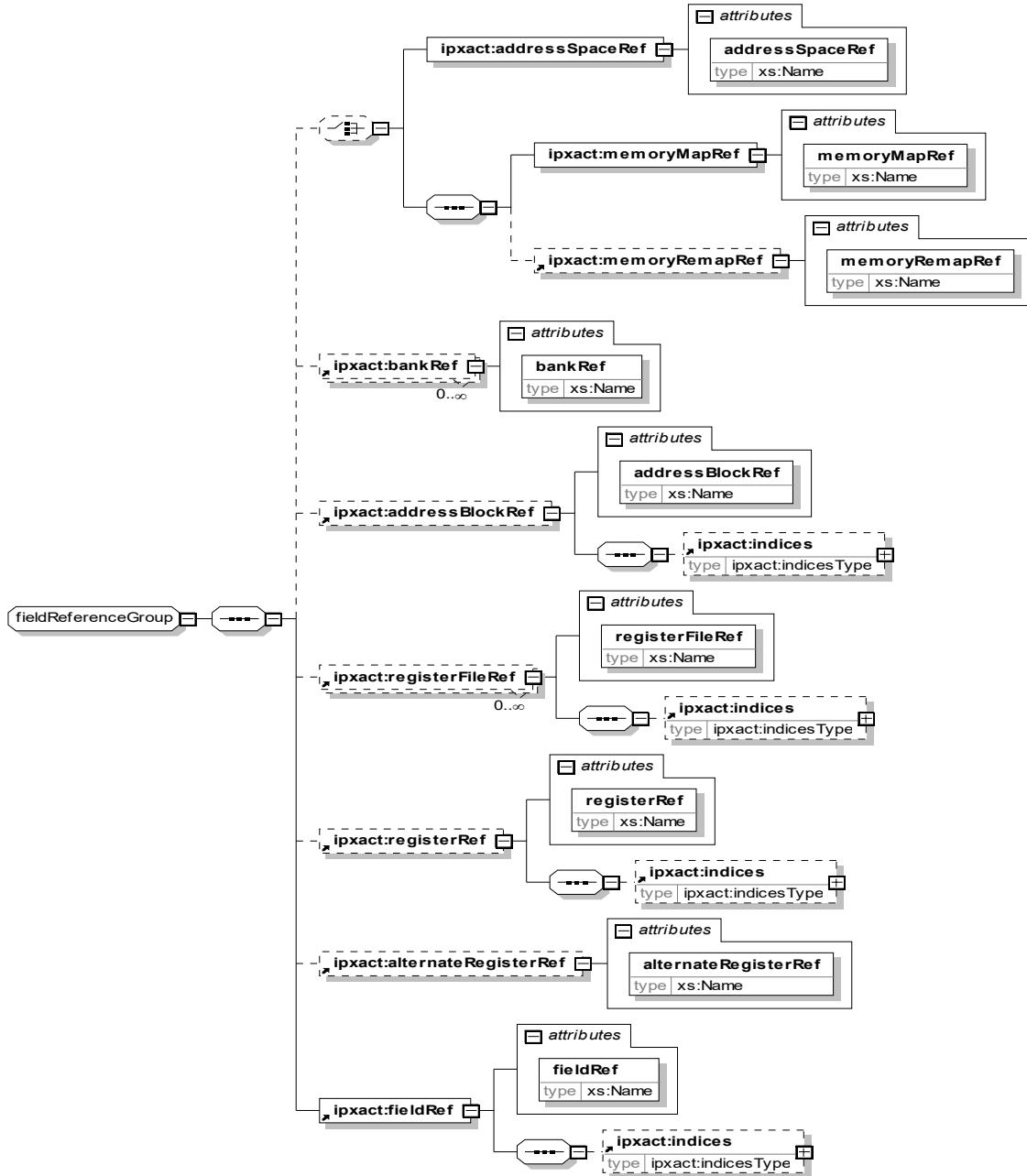
- *Little endian (little)* means the least significant byte of any multi-byte data field is stored at the lowest memory address, which is also the address of the larger field.

NOTE—The description of endianness is byte-centric as that is the most common least addressable unit (LAU). However, this description generally applies to any size LAU.

C.13 fieldReferenceGroup

C.13.1 Schema

The following schema details the information contained in the **fieldReferenceGroup**.



C.13.2 Description

The **fieldReferenceGroup** references another field in the encapsulating component of this group using the sub-elements listed below. These sub-elements describe the relative hierarchical path of a field in the

component. The **fieldReferenceGroup** contains the following elements:

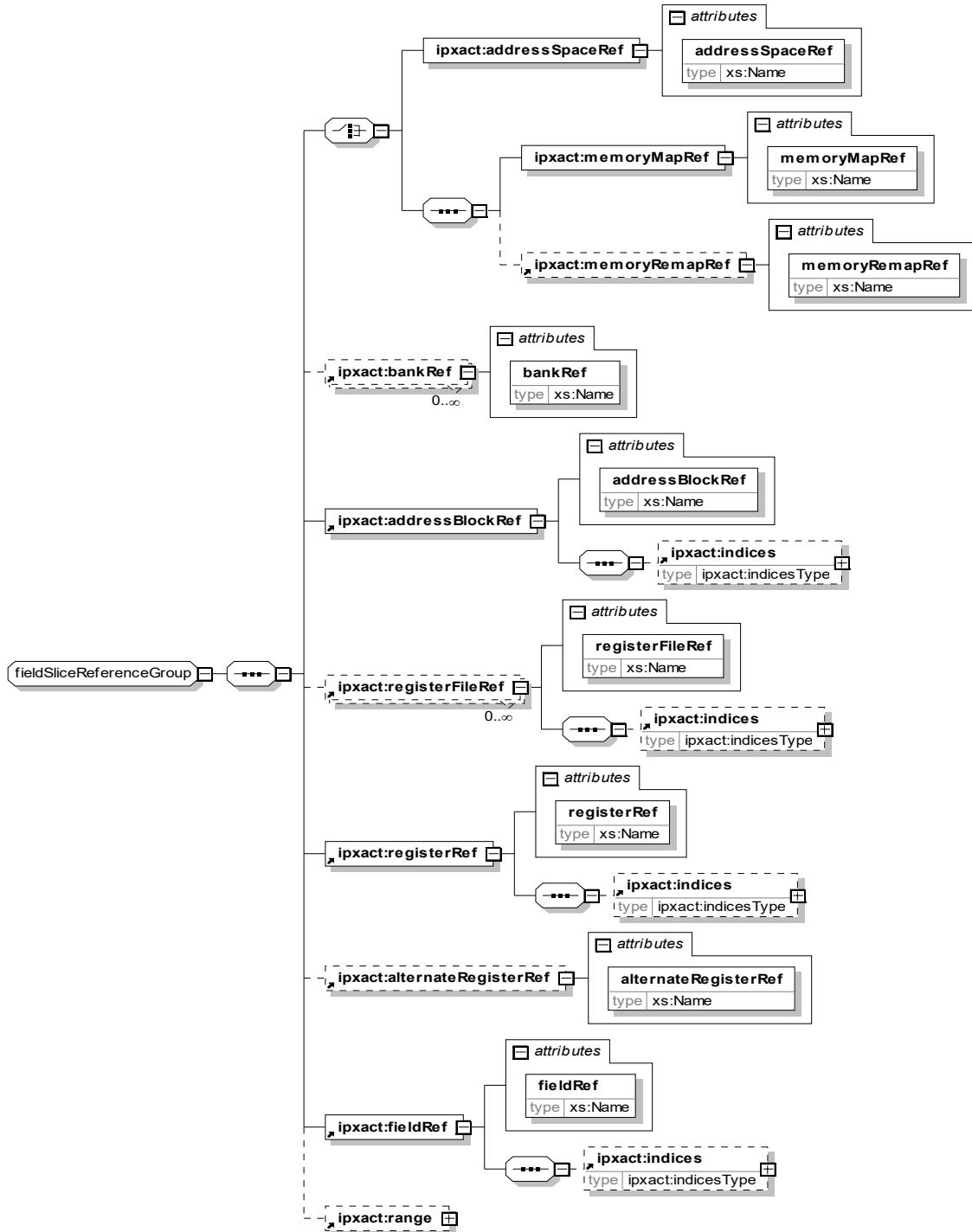
- a) **addressSpaceRef** (optional) references an addressSpace by name if the other field is located in a local memory map of this address space. The local memory map is implicitly referenced.
- b) **memoryMapRef** (optional) references a memoryMap by name if the other field is located in this memory map.
- c) **memoryRemapRef** (optional) references a memoryRemap in the memoryMap referenced by memoryMapRef if this other field is located in this memoryRemap.
- d) **bankRef** (optional) references a bank by name if the other field is located in this bank. The sequence of bank references indicates the bank hierarchy in the memory map.
- e) **addressBlockRef** (optional) references an addressBlock by name if the other field is located in this address block.
- f) **registerFileRef** (optional) references a registerFile by name if the other field is located in this register file. The sequence of register file references indicates the register file hierarchy in the address block.
- g) **registerRef** (optional) references a register by name if the other field is located in this register.
- h) **alternateRegisterRef** (optional) references an alternateRegister by name if the other field is located in this alternate register.
- i) **fieldRef** (mandatory) references a field by name to which the reference applies.
- j) **indices** elements can be used to select an array element if the encapsulated object references an array. If no indices element is present then the whole array is referenced. The total number of field elements referenced in the **fieldReferenceGroup** shall match the total number of field elements in the referencing field such that both can be linearized and the elements of the referencing and referenced field can be paired by the linear index value. See [SCR 1.43](#).

References to fields in memory remaps are not conditioned by modes referenced in memory remaps. For example, a **broadcastTo** element referencing a field in a memory remap implies a broadcast to that field even when the memory remap is not active.

C.14 fieldSliceReferenceGroup

C.14.1 Schema

The following schema details the information contained in the *fieldSliceReferenceGroup* element.



C.14.2 Description

The **fieldSliceReferenceGroup** references another field slice in the encapsulating component of this group using the sub-elements **addressSpaceRef**, **memoryMapRef**, **bankRef**, **addressBlockRef**, **registerFileRef**, **registerRef**, **alternateRegister**, and **fieldRef**. These elements describe the absolute hierarchical path of a field in the component.

The **fieldSliceReferenceGroup** definition contains the following elements:

- a) **addressSpaceRef** (mandatory) references an **addressSpace** by name if the other field is located in a local memory map of this address space. The local memory map is implicitly referenced.
- b) **memoryMapRef** (mandatory) references a **memoryMap** by name if the other field is located in this memory map.
- c) **memoryRemapRef** (optional) references a **memoryRemap** in the **memoryMap** referenced by **memoryMapRef** if this other field is located in this **memoryRemap**.
- d) **bankRef** (optional) references a **bank** by name if the other field is located in this bank. The sequence of bank references indicates the bank hierarchy in the memory map.
- e) **addressBlockRef** (optional) references an **addressBlock** by name if the other field is located in this address block.
- f) **registerFileRef** (optional) references a **registerFile** by name if the other field is located in this register file. The sequence of register file references indicates the register file hierarchy in the address block.
- g) **registerRef** (mandatory) references a **register** by name if the other field is located in this register.
- h) **alternateRegisterRef** (optional) references an **alternateRegister** by name if the other field is located in this alternate register.
- i) **fieldRef** (mandatory) references a **field** by name to which the reference applies.
- j) **range** (optional) references a slice from the referenced field.

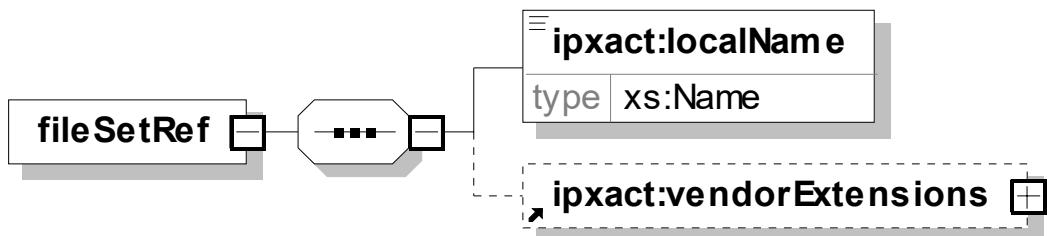
The **indices** elements can be used to select an array element if the encapsulated object references an array.

References to field slices in memory remaps are not conditioned by modes referenced in memory remaps. For example, a **fieldMap** element referencing a field slice in a memory remap implies access to that field slice from the port encapsulating the **fieldMap** element even when the memory remap is not active.

C.15 fileSetRef

C.15.1 Schema

The following schema details the information contained in the **fileSetRef** element.



C.15.2 Description

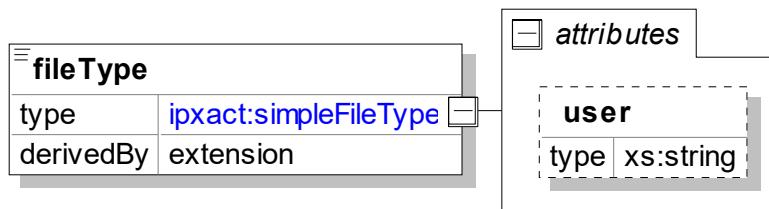
The **fileSetRef** element defines a reference to a **fileSet** contained in the containing document. The **fileSetRef** element contains the following element:

- a) **localName** (mandatory; type: *Name*) shall contain a name of a **fileSet/name** within the local description.
- b) **vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See [C.27](#).

C.16 fileType

C.16.1 Schema

The following schema details the information contained in the *fileType* element.



C.16.2 Description

The **fileType** element defines the format of a referenced file. The **fileType** element contains one or more of the following elements and attributes:

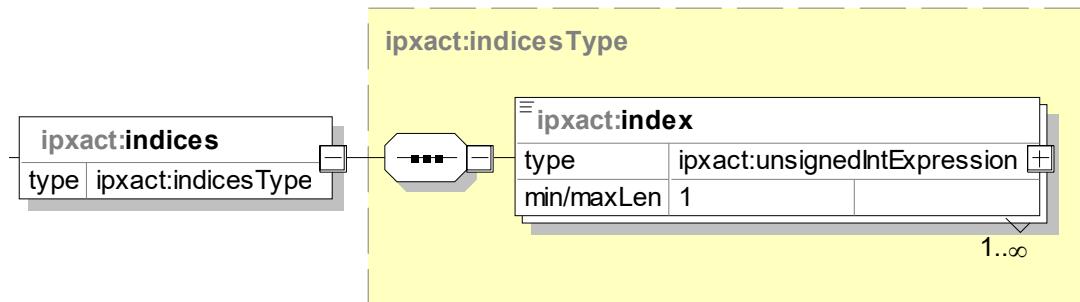
- a) **fileType** (mandatory) describes the type of file referenced from this enumerated list of industry standard files types.
 - 1) **unknown**
 - 2) **asmSource**, **cSource**, **cppSource**, **eSource**, **OVASource**, **perlSource**, **pslSource**, **SVASource**, **tclSource**, **veraSource**, **systemCSource**, **systemCSource-2.0**, **systemCSource-2.0.1**, **systemCSource-2.1**, **systemCSource-2.2**, **systemVerilogSource**, **systemVerilogSource-3.0**, **systemVerilogSource-3.1**, **systemVerilogSource-3.1a**, **verilogSource**, **verilogSource-95**, **verilogSource-2001**, **verilogSource-2005**, **vhdlSource**, **vhdlSource-87**, **vhdlSource-93**, **vhdlSource2002**, **vhdlSource2008**
 - 3) **swObject** and **swObjectLibrary**
 - 4) **vhdlBinaryLibrary** and **verilogBinaryLibrary**
 - 5) **executableHdl** and **unelaboratedHdl**
 - 6) **systemVerilogSource**, **systemVerilogSource-3.0**, **systemVerilogSource-3.1**, **systemVerilogSource-3.1a**, **systemVerilogSource-2009**, **systemVerilogSource-2012**, and **systemVerilogSource-2017**
 - 7) **systemCSource**, **systemCSource-2.0**, **systemCSource-2.0.1**, **systemCSource-2.1**, **systemCSource-2.2**, and **systemCSource-2.3**

- 8) **systemCBinaryLibrary**
 - 9) **veraSource**
 - 10) **eSource**
 - 11) **perlSource**
 - 12) **tclSource**
 - 13) **OVASource**
 - 14) **SVASource**
 - 15) **pslSource**
 - 16) **SDC**
 - 17) **vhdlAmsSource**
 - 18) **verilogAmsSource**
 - 19) **systemCAMSOURCE**
 - 20) **libertySource**
 - 21) **spiceSource**
 - 22) **systemRDL**, **systemRDL-1.0**, and **systemRDL-2.0**
 - 23) **user**
- b) **user** (optional; type: *string*) attribute indicates a user-defined file type name valid only when **fileType** is **user**.

C.17 indices

C.17.1 Schema

The following schema details the information contained in the **indices** element.



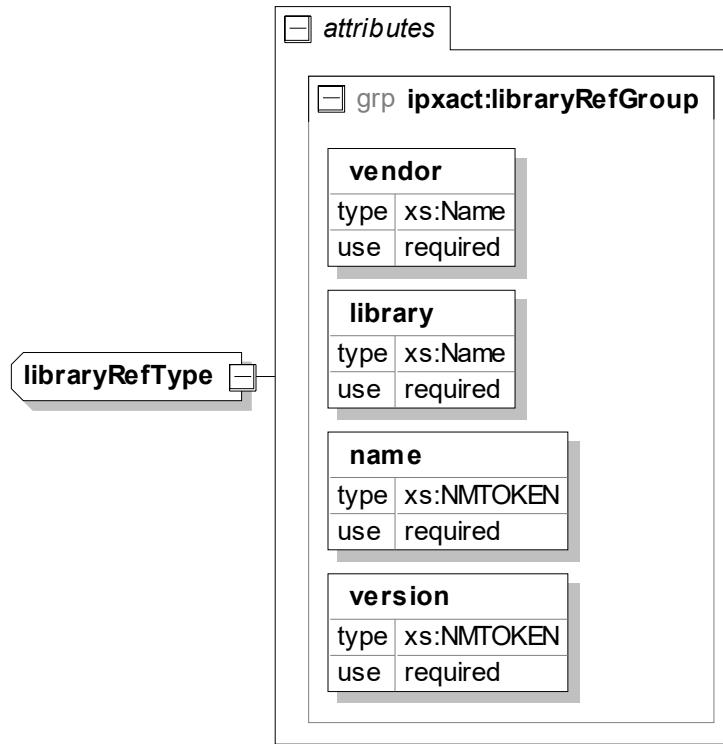
C.17.2 Description

The **indices** element specifies a multi-dimensional list of **index** elements. An **index** (type **unsignedIntExpression**, see [C.7.11](#)) is a multi-dimensional array and follows C-semantics for indexing. The **indices** element is typically used to reference into an arrayed/vectored element, such as a port.

C.18 libraryRefType

C.18.1 Schema

The following schema details the information contained in the *libraryRefType* element.



C.18.2 Description

The *libraryRefType* element defines a set of four attributes that reference another IP-XACT description through the unique VLVN identifier.

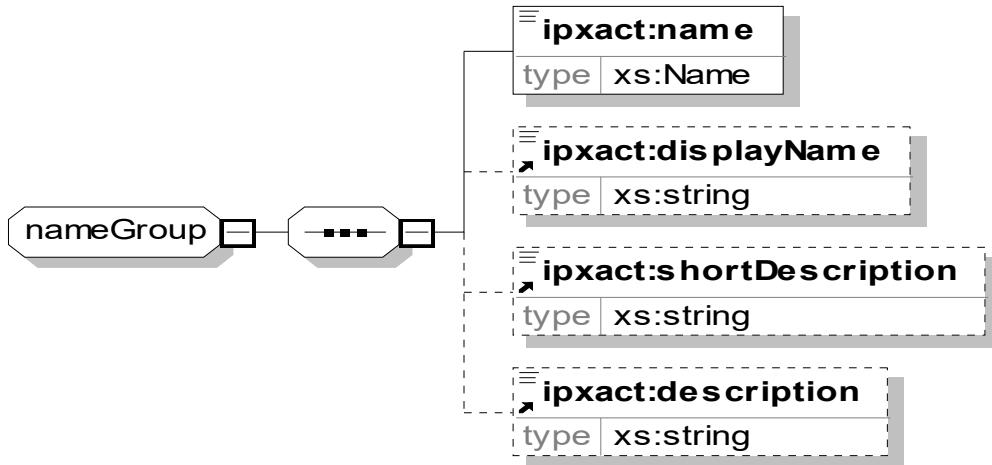
- a) The **vendor** attribute (mandatory; type: *Name*) identifies the owner of the referenced description.
- b) The **library** attribute (mandatory; type: *Name*) identifies the library of the referenced description.
- c) The **name** attribute (mandatory; type: *NMTOKEN*) identifies the name of the referenced description.
- d) The **version** attribute (mandatory; type: *NMTOKEN*) identifies the version of the referenced description.

C.19 Name groups

C.19.1 nameGroup group

C.19.1.1 Schema

The following schema details the information contained in the **nameGroup** group.



C.19.1.2 Description

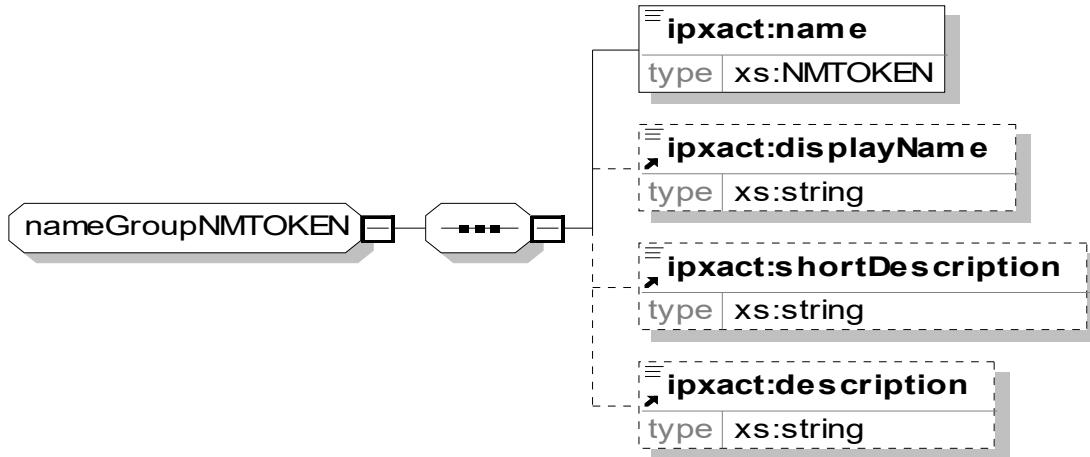
The **nameGroup** group defines any descriptive text for the containing element. The **nameGroup** group definition contains the following elements:

- a) **name** (mandatory; type: *Name*) identifies the containing element.
- b) **displayName** (optional; type: *string*) allows a short descriptive text to be associated with the containing element. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- c) **shortDescription** (optional; type: *string*) is a more compact textual description of the containing element compared to **description**.
- d) **description** (optional; type: *string*) allows a textual description of the containing element.

C.19.2 nameGroupNMOKEN group

C.19.2.1 Schema

The following schema details the information contained in the **nameGroupNMOKEN** group.



C.19.2.2 Description

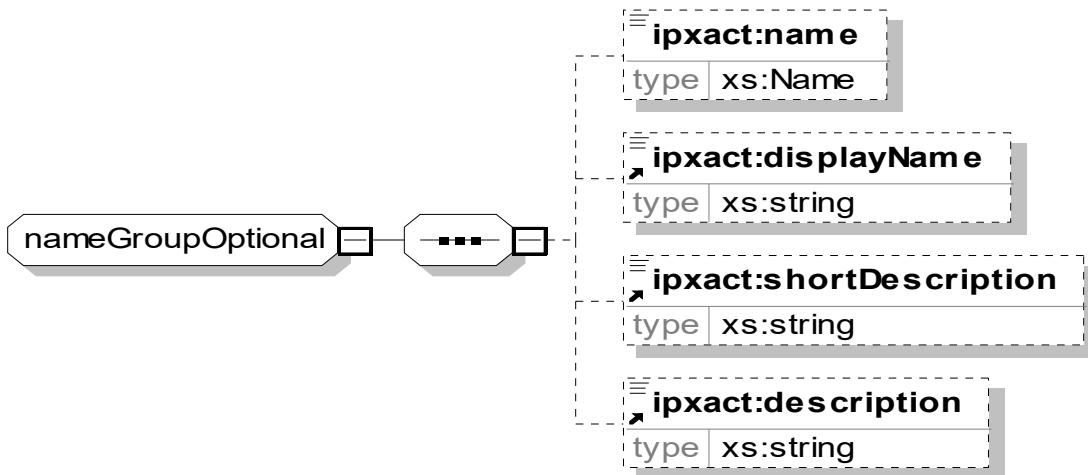
The **nameGroupNMOKEN** group defines any descriptive text for the containing element. The **nameGroupNMOKEN** group definition contains the following elements:

- name** (mandatory; type: *NMOKEN*) identifies the containing element. The name used shall match the corresponding port name found in any **views** of the containing component.
- displayName** (optional; type: *string*) allows a short descriptive text to be associated with the containing element. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- shortDescription** (optional; type: *string*) is a more compact textual description of the containing element compared to **description**.
- description** (optional; type: *string*) allows a textual description of the containing element.

C.19.3 nameGroupOptional group

C.19.3.1 Schema

The following schema details the information contained in the *nameGroupOptional* group.



C.19.3.2 Description

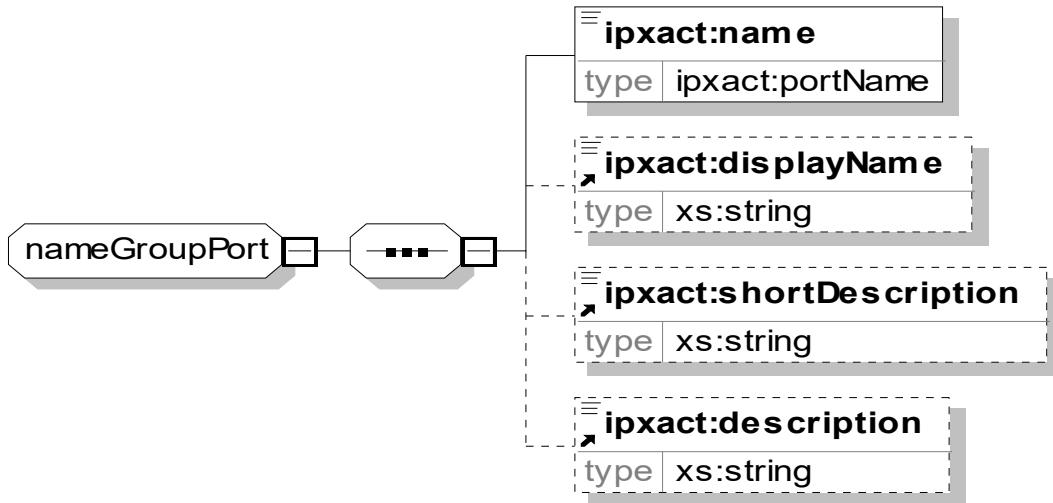
The *nameGroupOptional* group defines any descriptive text for the containing element. The *nameGroupOptional* group definition contains the following elements:

- a) **name** (optional; type: *Name*) identifies the containing element.
- b) **displayName** (optional; type: *string*) allows a short descriptive text to be associated with the containing element. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- c) **shortDescription** (optional; type: *string*) is a more compact textual description of the containing element compared to **description**.
- d) **description** (optional; type: *string*) allows a textual description of the containing element.

C.19.4 nameGroupPort group

C.19.4.1 Schema

The following schema details the information contained in the *nameGroupPort* group.



C.19.4.2 Description

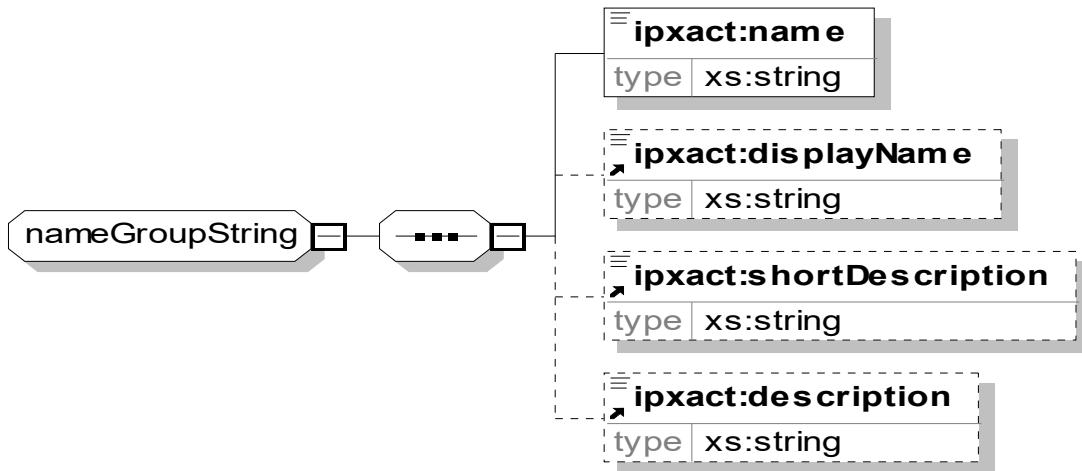
The *nameGroupPort* group defines any descriptive text for the containing element. The *nameGroupPort* group definition contains the following elements:

- name** (mandatory; type: *portName*) identifies the containing element.
- displayName** (optional; type: *string*) allows a short descriptive text to be associated with the containing element. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- shortDescription** (optional; type: *string*) is a more compact textual description of the containing element compared to **description**.
- description** (optional; type: *string*) allows a textual description of the containing element.

C.19.5 nameGroupString group

C.19.5.1 Schema

The following schema details the information contained in the *nameGroupString* group.



C.19.5.2 Description

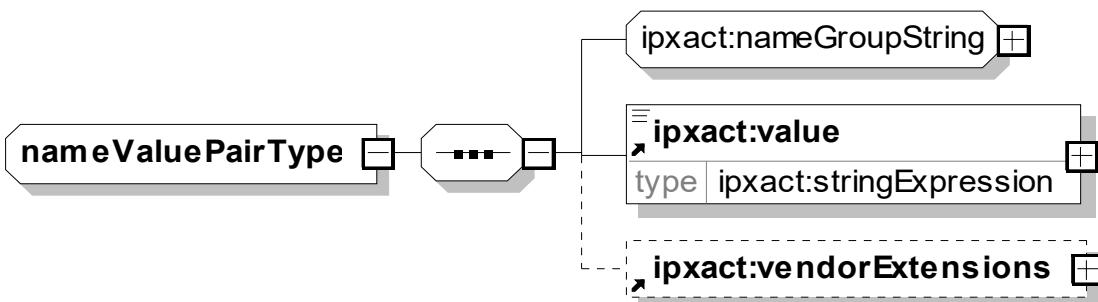
The *nameGroupString* group defines any descriptive text for the containing element. The *nameGroupString* group definition contains the following elements:

- name** (mandatory; type: *string*) identifies the containing element.
- displayName** (optional; type: *string*) allows a short descriptive text to be associated with the containing element. It is a string that can be used as an identifier outside of IP-XACT documents, for instance, in human-readable documents or graphical user interfaces. It does not have to be unique.
- shortDescription** (optional; type: *string*) is a more compact textual description of the containing element compared to **description**.
- description** (optional; type: *string*) allows a textual description of the containing element.

C.20 nameValuePairType

C.20.1 Schema

The following schema details the information contained in the *nameValuePairType* element.



C.20.2 Description

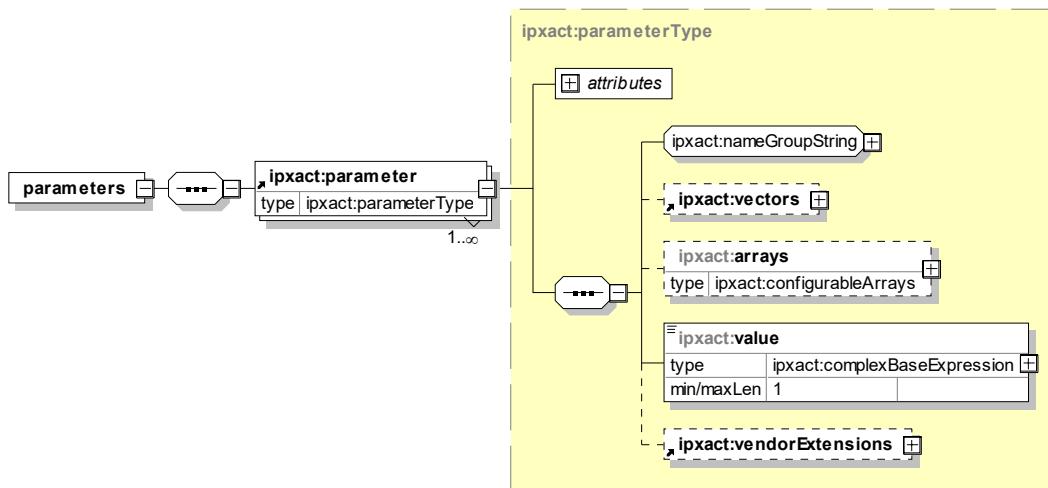
The ***nameValuePairType*** element specifies the name and value type used for resolvable elements. The ***nameValuePairType*** element contains the following elements:

- nameGroupString*** group is defined in [C.19.5](#).
- value*** (mandatory; type: ***stringExpression***, see [C.7.5](#)) contains the actual value of the **parameter**.
- vendorExtensions*** (optional) adds any extra vendor-specific data related to the port. See [C.27](#).

C.21 parameters

C.21.1 Schema

The following schema details the information contained in the **parameters** element.



C.21.2 Description

The **parameters** element contains an unbounded list of **parameter** elements. **parameter** (mandatory) defines a configurable element related to the containing element. The parameter definition allows for the assignment of a **name**, **parameterId**, **vectors**, **arrays**, and a **value**. The **parameter** element also allows for vendor attributes to be applied. The **parameter** element definition contains the following elements:

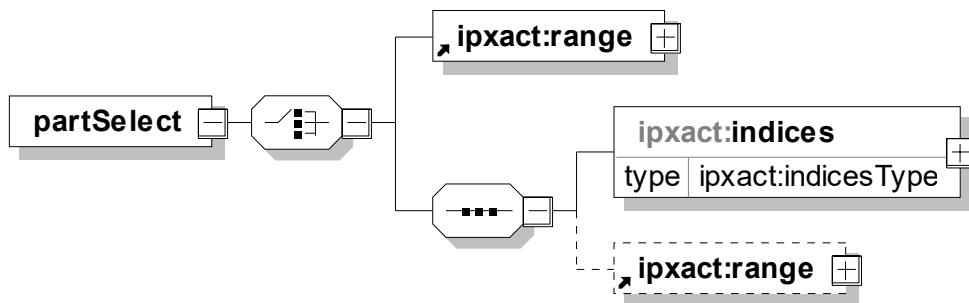
- attributes*** specify the parameter behavior, like ***type***, ***sign***, ***parameterId***, etc. See [C.6](#).
- nameGroupString*** is defined in [C.19.5](#).
- vectors*** (optional) specifies the dimensions of a vector when the parameter represents a packed array or bit-vector (where the ***type*** is **bit**). See [C.26.2](#).
- arrays*** (optional; type: ***configurableArrays***, see [C.4.2](#)) specifies dimensions for a parameter defined as an array.
- value*** (mandatory; type: ***complexBaseExpression***, see [C.7](#)) contains the actual value of the **parameter**. The ***value*** element is further constrained by the ***type***, ***sign***, and ***array*** properties specified on the **parameter** element.
- vendorExtensions*** (optional) adds any extra vendor-specific data related to the **parameter**. See [C.27](#).

See also [SCR 5.2](#), [SCR 5.16](#), [SCR 5.20](#), [SCR 5.21](#), and [SCR 5.22](#).

C.22 partSelect

C.22.1 Schema

The following schema details the information contained in the **partSelect** element.



C.22.2 Description

The **partSelect** element defines a selection of subelements from within a referenced port element. A part selection can consist of either a single range specification or multiple index specifications along with an optional range specification.

- a) **range** defines the range of bits or array elements being referenced. If **index** elements are also present, the **range** applies after all **index** elements. See [C.25](#).
- b) **indices** specifies a list of **index** elements. The **indices** specify an element in the IP-XACT object to which this **partSelect** applies. See [C.17](#).

When multiple indices are present, they correspond to the dimensions of the defined port, account for array ranges first, and then account for vector ranges. The **partSelect** element indices shall be a proper index in the multi-dimensional port, and the **partSelect** range shall select consecutive bits from that port.

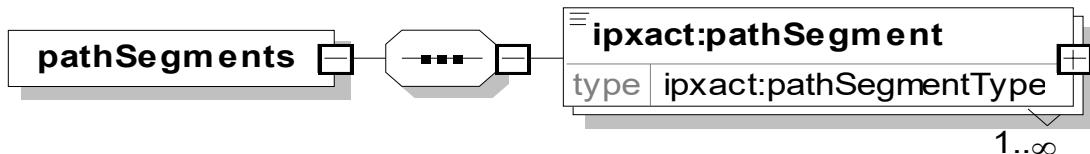
Also see [SCR 6.55](#) and [SCR 6.56](#).

C.23 pathSegments

C.23.1 pathSegment

C.23.1.1 Schema

The following schema details the information contained in the **pathSegments** element, which occurs in **simpleAccessHandle** and **slicedAccessHandle**.



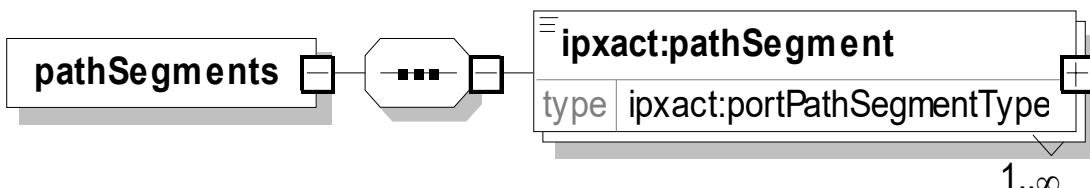
C.23.1.2 Description

The **pathSegments** element specifies an ordered list of **pathSegment** elements. A **pathSegment** is one node in the hierarchical path. When concatenated with a desired separator, the elements in this form a language-specific path for the parent slice into the referenced view. The **pathSegment** element is of type **unresolvedStringExpression**.

C.23.2 portPathSegment

C.23.2.1 Schema

The following schema details the information contained in the **pathSegments** element which occurs in **portAccessHandles**.



C.23.2.2 Description

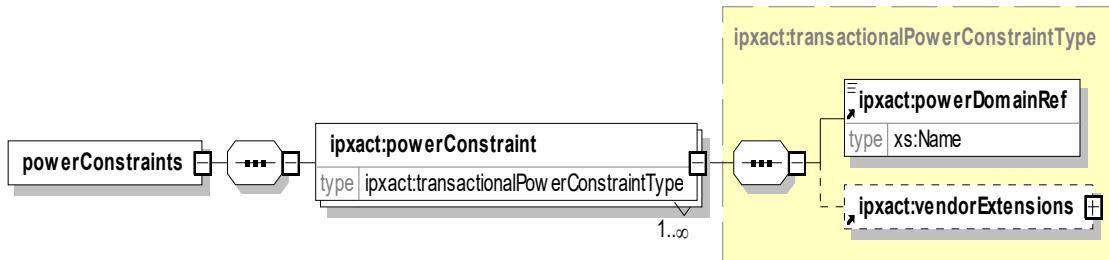
The **pathSegments** element specifies an ordered list of **pathSegment** elements. A **pathSegment** is one node in the hierarchical path. When concatenated with a desired separator, the elements in this form a language-specific path for the parent slice into the referenced view. The **pathSegment** element is of type **stringExpression**.

C.24 Power constraints

C.24.1 transactionalPowerConstraints

C.24.1.1 Schema

The following schema details the information contained in the transactional **powerConstraints** element.



C.24.1.2 Description

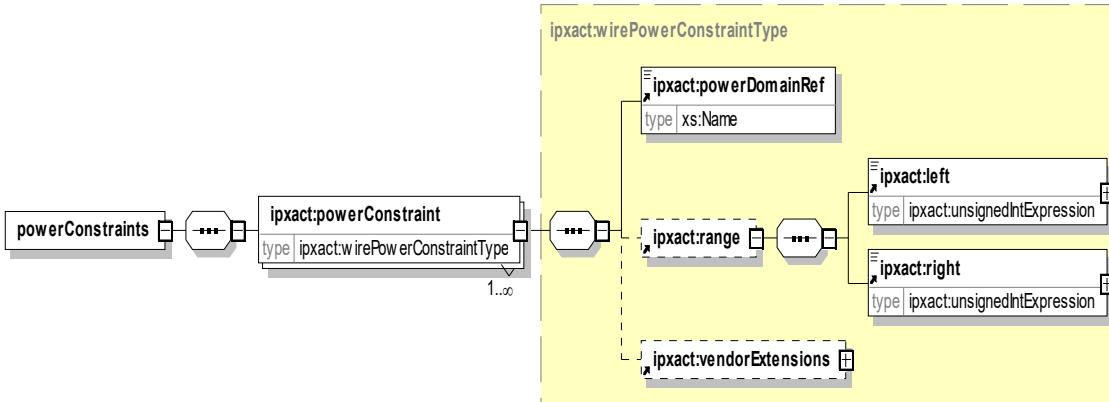
The transactional **powerConstraint** element describes the ports power domain. The **powerConstraints** element definition contains the following elements:

- powerDomainRef** (mandatory) references a power domain of the encapsulating component.
- vendorExtensions** (optional) contains any extra vendor-specific data related to the transactional **powerConstraint**. See [C.27](#).

C.24.2 wirePowerConstraints

C.24.2.1 Schema

The following schema details the information contained in the wire **powerConstraints** element.



C.24.2.2 Description

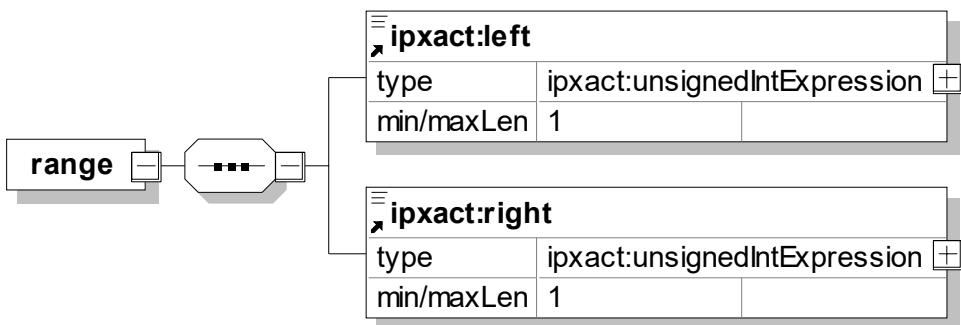
The wire **powerConstraint** element describes the ports power domain. The wire **powerConstraint** element definition contains the following elements:

- a) **powerDomainRef** (mandatory) references a power domain of the encapsulating component.
- b) **range** (optional) defines the range of bits or array elements being referenced. The range element contains the following elements:
 - 1) **left** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the left range for the bit slice from a parameter or port.
 - 2) **right** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the right range for the bit slice from a parameter or port.
- c) **vendorExtensions** (optional) contains any extra vendor-specific data related to the wire **powerConstraint**. See [C.27](#).

C.25 range

C.25.1 Schema

The following schema details the information contained in the **range** element.



C.25.2 Description

The **range** element defines the range of bits or array elements being referenced. The **range** element contains the following elements:

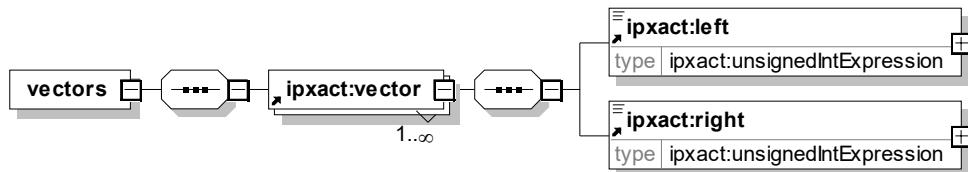
- a) **left** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the left range of the bit slice.
- b) **right** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the right range of the bit slice.

C.26 Vectors

C.26.1 vectors

C.26.1.1 Schema

The following schema details the information contained in the **vectors** element.



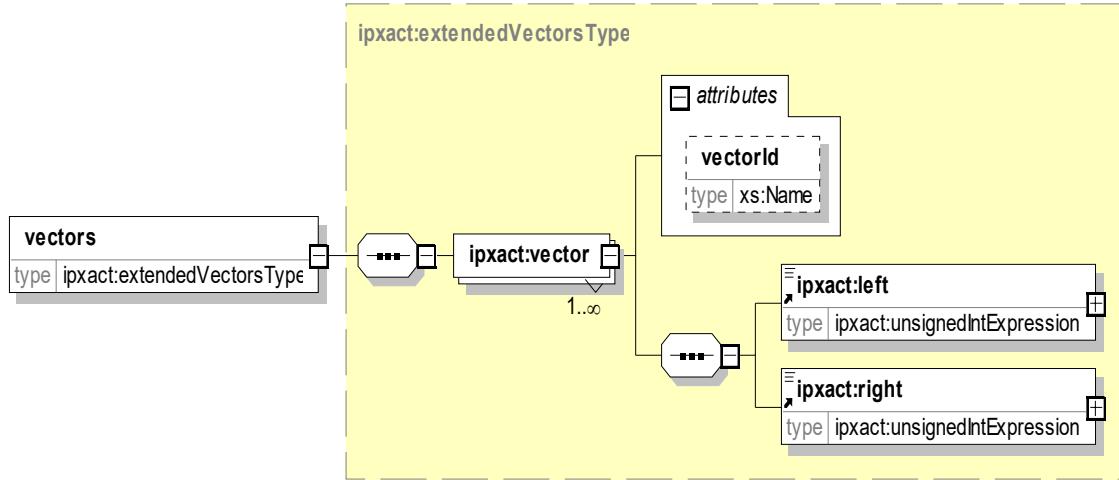
The **vectors** element specifies the dimensions of a **vector** when the encapsulating parameter or port represents a packed array or bit-vector (where the **type** is **bit**). The **vector** element contains the following attributes and elements:

- left** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the left range for the bit slice from a parameter or port.
- right** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the right range for the bit slice from a parameter or port.

C.26.2 extendedVectorsType

C.26.2.1 Schema

The following schema details the information contained in the **extendedVectorsType** element.



The **vectors** element specifies the dimensions of a **vector** when the encapsulating parameter or port represents a packed array or bit-vector (where the **type** is **bit**). The **vector** element contains the following attributes and elements:

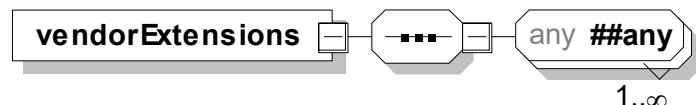
- vectorId** (optional; type: *Name*) identifies the vector element for reference in the constrained attribute. See [6.15.11.2](#).

- b) **left** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the left range for the bit slice from a parameter or port.
- c) **right** (mandatory; type: *unsignedIntExpression*, see [C.7.12](#)) specifies the right range for the bit slice from a parameter or port.

C.27 vendorExtensions

C.27.1 Schema

The following schema details the information contained in the **vendorExtensions** element.



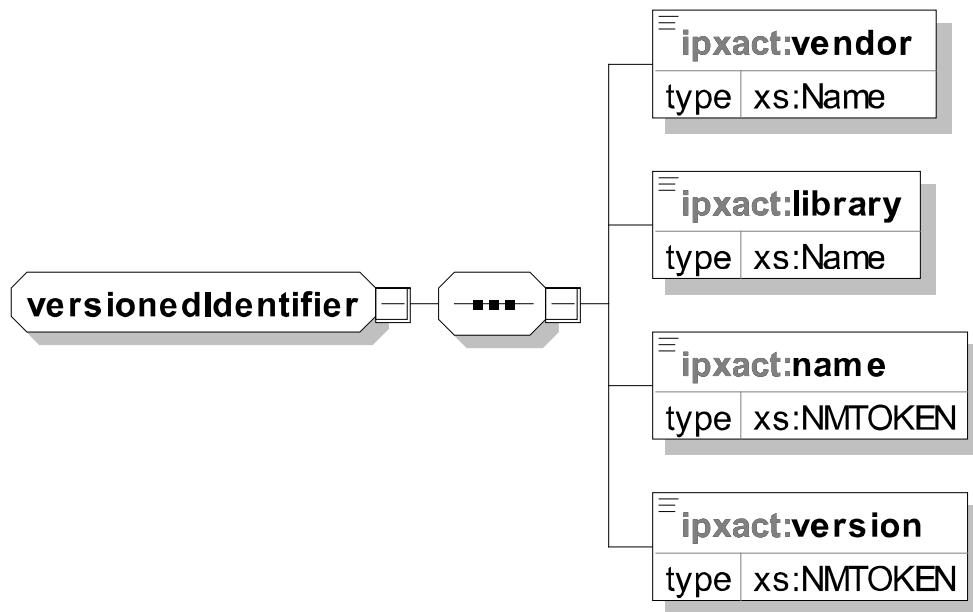
C.27.2 Description

The **vendorExtensions** element is a place in the description in which any vendor-specific information can be stored. The **vendorExtensions** element allows any well-formed description.

C.28 versionedIdentifier group

C.28.1 Schema

The following schema details the information contained in the **versionedIdentifier** group.



C.28.2 Description

The **versionedIdentifier** group defines a unique reference of or from an IP-XACT description. Only one object with a given VLVN may be present in a DE at any given time. The timing and way to change the VLVN of an object is completely up to the user or developer. The **versionedIdentifier** group definition contains the following four subelements:

- a) **vendor** (mandatory; type: *Name*) identifies the owner of this description. The format of the **vendor** element is the company internet domain name in left-to-right order (e.g., `accellera.org` not `org.accellera`).
- b) **library** (mandatory; type: *Name*) identifies the library of this description. This allows a vendor to group descriptions.
- c) **name** (mandatory; type: *NMTOKEN*) identifies the name of this description within a library.
- d) **version** (mandatory; type: *NMTOKEN*) identifies the version of this description. This allows a vendor to provide many descriptions that all have the same name, but are still uniquely identified. The **version** may appear as an alphanumeric string and contain a set of substrings, with non-alphanumeric delimiters in-between. Each IP supplier shall have their own cataloguing system for setting version numbers.

See also [SCR 1.1](#) and [SCR 1.2](#).

C.28.3 Sorting and comparing version elements

Sorting and comparing **version** elements determines whether

- An IP is a component that has been previously imported.
- Multiple versions of the same IP exist in a design.
- A newer version of an IP exists.

To sort and compare **version** elements, subdivide the version number into major fields and subfields. Major fields may be separated by a non-alphanumeric delimiter such as `.`, `-`, `_`, etc. Each major field can be compared to determine equivalence and broken down further into subfields if necessary.

C.28.3.1 Comparison rules

- a) Each **version** element is broken into its major fields, which are separated using the appropriate delimiter (e.g., `:` or `.`).
- b) Major fields are compared against each other from left to right.
- c) Subfields, within each major field, need to be examined if the major fields are alphanumeric. Each major field shall have alphabetical and numerical subfields that are separated from right to left.
- d) To summarize the rules for the comparison of each subfield in a major field:
 - 1) Numeric—Compare the integer values of numeric subfields.
 - 2) Alphabetic
 - i) String—Perform a simple string comparison.
 - ii) Case—Ignore alphabetic case (e.g., `a` and `A` are the same).

It is possible for different representations of version numbers to be considered equal. For example, under these rules, `A1` and `A01` are equal, since numerical subfields are compared numerically, and `A.B` equals `A_B`, since delimiters are not compared.

C.28.3.2 Comparison examples

The following examples illustrate the sorting and comparing of a **version** elements:

Example 1

The first case uses: 205:75WR16 and 215:50HR15.

- a) Each of these version numbers break down into the following two major fields, separated by the : delimiter: 205 75WR16 and 215 50HR15.
- b) Major fields are compared against each other from left to right. In this example, the first major fields (205 and 215) differ between the VLVN strings and the comparison ends there. This case is also simplified since the first major field is an integer (i.e., numeric).
- c) Subfields, within each major field, need to be examined if the major fields are alphanumeric. Each major field shall have alphabetical and numerical subfields that are separated from right to left.

Example 2

In the next case, two VLVN have the same first major field, and their second major subfields need to be compared: e.g., 205:45R16 and 205:55R15.

- a) The first major field (205) is the same between these two VLVN, so the second major field is checked. These second major fields are broken down into the following alphabetic and numeric subfields: 45 R 16 and 55 R 15.
- b) The subfields are compared from left to right. The first (and in this case only) comparison is 45 versus 55, so these subfields are not equal. The major fields are not equivalent.

C.28.4 Version control

Each file conforming to the top-level schema has a set of VLVN elements that, when considered together, form a unique identifier (a *version control number*) for the information contained in that XML document. The VLVN of any IP-XACT information is not the same as the version of the file that might contain that information.

NOTE—An XML file might be revised in a way that does not materially affect the IP-XACT information content. For example, copyright notices are updated, comments are added, and environment variable names used as part of the filenames might be changed (but still point to the same files). These changes may not necessitate changing the VLVN.

Many developers of IP libraries use a version control system to track updates and changes to the various files that contribute to the overall design and IP package information. At any time, individual files may be modified and updated as development of that design or IP progresses. At appropriate junctures, releases are made, each consisting of a particular combination of files at different levels of a version.

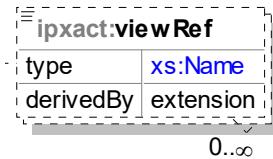
An IP-XACT description is one of the files that can be very usefully tracked in this way and updated in line with other design modifications. There is no direct link between the version number of the file and the VLVN identifier contained in that description. In many cases, however, the VLVN can be coordinated with the overall release package version.

See also [SCR 1.1](#) and [SCR 1.2](#).

C.29 viewRef

C.29.1 Schema

The following schema details the information contained in the **viewRef** element.



C.29.2 Description

The occurrence of **viewRef** (type: *Name*) elements (zero to more) in an **accessHandle**, **abstractionType**, **driver**, **wireTypeDef**, **domainTypeDef**, **signalTypeDef**, **transTypeDef**, and **structPortTypeDef** elements shall meet the following rules and interpretation:

- If an **accessHandle**, **abstractionType**, **wireTypeDef**, **domainTypeDef**, **signalTypeDef**, **transTypeDef**, or **structPortTypeDef** element does not have **viewRef** elements, then all existing component view elements are referenced implicitly. This implies there is only one **accessHandle**, **abstractionType**, **wireTypeDef**, **domainTypeDef**, **signalTypeDef**, **transTypeDef**, or **structPortTypeDef** element in the enclosing **accessHandles**, **abstractionTypes**, **wireTypeDefs**, **domainTypeDefs**, **signalTypeDefs**, **transTypeDefs**, and **structPortTypeDefs** elements.
- A view name may be referenced once in all **viewRef** elements of the enclosing **accessHandles**, **abstractionTypes**, **wireTypeDefs**, **domainTypeDefs**, **signalTypeDefs**, **transTypeDefs**, or **structPortTypeDefs** elements.
- If a view name is not explicitly or implicitly referenced in a **driver**, **wireTypeDefs**, **transTypeDefs**, or **structPortTypeDefs** elements, the enclosing element of the **wireTypeDefs**, or **transTypeDefs**, or **structPortTypeDefs** element does not exist in that view.

C.30 xml:id

The **xml:id** attribute can be specified to provide a unique identifier for the element. Refer to *xml:id Version 1.0 [xml-ref]*. The attribute value is required to be unique in the document. The attribute is not referenced by any of the features in the IP-XACT language. **xml:id** has been provided so tools can more precisely identify nodes in the document to provide, e.g., better change control or to attach documentation and other related information to the node in the document.

xml:id is available on all nodes that otherwise cannot be uniquely identified, i.e., all items that can have multiple occurrences at the same level of the hierarchy.

C.31 Component vs. Abstraction Definition Ports

[Table C.2](#) demonstrates when component versus abstraction definition port default values apply.

Table C.2—Component vs. Abstraction Definition

Component port has default value	Component port has port map in component bus interface with default value on logical port	Design has adhoc to component port with tiedValue default	Design has interconnection to component bus interface and logical port is not connected	Default value that applies
no	no	x	x	none
yes	no	yes	x	component port default value
no	yes	x	yes	abstraction definition port default value
yes	yes	no	no	none
yes	yes	yes	no	component port default value
yes	yes	no	yes	abstraction definition port default value
yes	yes	yes	yes	component port default value

Annex D

(normative)

Types

Many elements and attributes defined in the standard have associated types. These types define the legal values and ranges for input into these element and attributes.

D.1 boolean

The **boolean** type defines two possible values, **true** and **false**.

D.2 float

The **float** type defines a decimal floating point number based on the IEEE single-precision 32-bit floating point type (see IEEE Std 754-1985 [\[B2\]](#)).

D.3 ID or IDREF

The **ID** or **IDREF** type defines a unique identifier through the containing description. It needs to begin with a letter or underscore (_). An **ID** or **IDREF** shall contain only letters, numbers, and the underscore (_), dash (-), and dot (.) characters. Any leading or trailing spaces are removed.

D.4 instancePath

The **instancePath** type defines a series of **Name** type character strings (see [D.7](#)), separated by a slash (/). Any leading or trailing space is removed.

D.5 integer

The **integer** type defines a decimal integer number of infinite precision, containing the numbers 0–9.

D.6 libraryRefType

The **libraryRefType** type is an element type, not a data type. This type defines the four attributes of a VLVN required for a reference from one description to another description. See [C.18](#).

D.7 Name

The **Name** type defines a series of any characters, excluding embedded whitespace. It needs to begin with a letter, colon (:), or underscore (_). A **Name** shall contain only letters, numbers, and the colon (:), underscore (_), dash (-), and dot (.) characters. Any leading or trailing spaces are removed.

D.8 NMTOKEN

The **NMTOKEN** type defines a series of any characters, excluding embedded whitespace. It shall contain only letters, numbers, and the colon (:), underscore (_), dash (-), and dot (.) characters. Any leading or trailing spaces are removed.

D.9 NMTOKENS

The **NMTOKENS** type defines a series of any characters, including embedded whitespace. It shall contain only letters, numbers, and the colon (:), underscore (_), dash (-), and dot (.) characters.

D.10 portName

The **portName** type defines a series of any characters, excluding embedded whitespace. It shall contain only letters, numbers, and the colon (:), underscore (_), dash (-), and dot (.) characters. It also needs to begin with a letter, colon (:), or underscore (_). Any leading or trailing spaces are removed.

D.11 ipxactURI

The **ipxactURI** type defines a string of characters for an absolute or relative path to a file, a directory, or an executable in URI format (`xs:anyURI`), except it can contain environment variables in the `${ENV_VAR}` form, which are replaced by their value(s) to provide the underlying URI.

D.12 string

The **string** type defines a series of any characters and may include spaces.

D.13 token

The **token** type defines a series of any characters, excluding carriage-return, line-feed, and tab. Any leading or trailing spaces are removed, and all internal sequences of two or more spaces are reduced to one space.

Annex E

(normative)

SystemVerilog expressions

This standard utilizes *SystemVerilog expressions* as a means to specify an equation as the value of an element. Expressions can be specified for all predefined element values having an associated type (see [C.7](#) and [C.21](#)) that allows for an expression to be specified. Besides predefined types, parameters can also be specified using expressions, in which case the type attributes define the type of expression that can be specified.

E.1 Overview

The IP-XACT Expression Language is based on the SystemVerilog Expression Language as specified in IEEE Std 1800-2012.

The goal of the IP-XACT Expression Language is for it to be a subset of IEEE Std 1800-2012, i.e., the expressions defined in IP-XACT should be easily ported-to or incorporated-into a SystemVerilog file and interpreted by any SystemVerilog parser/processor.

The IP-XACT Expression Language, however, should also be able to represent constructs and functionality specific to the IP-XACT language. To make the language more in-line with the IP-XACT language, some functionality has been limited and some changes to the original SystemVerilog Expression Language were introduced.

E.2 Data-types

A subset of the SystemVerilog data-types are used by the IP-XACT Expression Language. All IP-XACT data-types map to a specific SystemVerilog data-type. The casting functionality and operators follow the same subset of the functionality specified by the SystemVerilog Expression Language.

It has been decided to not support complex, user-defined, and time data-types in IP-XACT; it has also been decided to support only two-state values and not allow for unknown or high-impedance values. Each supported System Verilog data type also has a additional corresponding unresolved data type in the IP-XACT Expression Language. The supported data-types are shown in [Table E.1](#).

E.2.1 bit data type

The **bit** data-type is equivalent to the **bit** data-type in SystemVerilog.

E.2.2 byte data type

The **byte** data-type is equivalent to the **byte** data-type in SystemVerilog, except it is not possible to specify an ASCII character and, therefore, ASCII characters cannot be assigned to a **byte** variable.

Table E.1—Data-types

Data-type	Description
bit	2-state data type, user-defined vector size
byte	2-state data type, 8-bit signed integer
shortint	2-state data type, 16-bit signed integer
int	2-state data type, 32-bit signed integer
longint	2-state data type, 64-bit signed integer
shortreal	32-bit floating point
real	64-bit floating point
string	An ordered collection of characters

E.2.3 shortint data type

The **shortint** data-type is equivalent to the **shortint** data-type in SystemVerilog.

E.2.4 int data type

The **int** data-type is equivalent to the **int** data-type in SystemVerilog.

E.2.5 longint data type

The **longint** data-type is equivalent to the **longint** data-type in SystemVerilog.

E.2.6 shortreal data type

The **shortreal** data-type is equivalent to the **shortreal** data-type in SystemVerilog.

E.2.7 real data type

The **real** data-type is equivalent to the **real** data-type in SystemVerilog.

E.2.8 string data type

The IP-XACT Expression Language **string** data-type is equivalent to the SystemVerilog **string** datatype; just like the SystemVerilog **string**, it supports only US-ASCII characters.

To simplify the string functionality and avoid potential issues with string-length, indexing, and casting to and from **long** or **bitstring** values, however, casting, indexing, and length tests are not supported. As a result, an IP-XACT **string** can be seen as an immutable, 'simple' object without length, which cannot be cast to other types or from other types. Also, only the string-equality, inequality, concatenation, replication, and inside operators are supported (see [Table E.2](#)).

E.2.9 Signed and unsigned data types

The IP-XACT data-types follow the same default signed/unsigned functionality as has been defined in the SystemVerilog Expression Language, i.e., the **bit** data-type is unsigned and the **byte**, **shortint**, **int**, and **longint** data-types are signed by default.

IP-XACT also allows specifying the signed-ness when assigning the values. and just like in SystemVerilog, the signed-ness can also be used when casting the value.

E.2.10 Unresolved data types

The IP-XACT unresolved data-types represent expressions which cannot be calculated into a single known value using only the information available in IP-XACT. Each supported System Verilog data-type has a corresponding unresolved data-type. Such unresolved data-types are propagated through the IP-XACT processing as unresolved expressions. An expression has an unresolved data-type value if it includes a function which returns a value with an unresolved data-type. Operators which have operands with values with unresolved data-types and functions which have arguments with values with unresolved data-types return a value with an unresolved data-type. These expressions can only be resolved to a single known value with additional information. An example is a pathSegment element which has a unresolvedStringExpression type and may contain a reference to an array index by including the \$ipxact_index_value() function. The actual value for the array index would be provided by a generator when the pathSegment needs to be resolved to a single known value.

E.3 Assignments

Variables and constants are fully declared and assigned using IP-XACT parameters (see [C.21](#)). In other words, the IP-XACT Expression Language itself does not support assignments and, therefore, does not support any assignment operators.

E.3.1 Single value assignment

To assign a single parameter with an identifier of `baseAddress` having a value of 'h100 in IP-XACT, use the following format:

```
<ipxact:parameter type="longint" parameterId="baseAddress">
  <ipxact:name>baseAddress</ipxact:name>
  <ipxact:value>'h100</ipxact:value>
</ipxact:parameter>
```

which is equivalent to the following SystemVerilog assignment:

```
longint baseAddress = 'h100;
```

E.3.2 Parameter type

The value of a parameter is itself an expression and should be specified accordingly. The type of the parameter expression is defined by the type attribute. Possible values for this type attribute are `bit`, `byte`, `shortint`, `int`, `longint`, `shortreal`, `real`, `string`. These types correspond to their SystemVerilog counterparts as described in [E.2](#).

The expression that is specified as the parameter's value is implicitly cast to the parameter's type. See [C.7](#) and [C.21](#).

E.3.3 Parameter signing

To change the signed-ness of a parameter, use the **sign** attribute (see [C.6.2](#)), whose value can be **signed** or **unsigned**, e.g.,

```
<ipxact:parameter type="longint" parameterId="baseAddress" sign="signed">
  <ipxact:name>baseAddress</ipxact:name>
  <ipxact:value>'h100</ipxact:value>
</ipxact:parameter>
```

This property influences only the following types: **bit**, **byte**, **shortint**, **int**, and **longint**:
 where the types **byte**, **shortint**, **int**, and **longint** are **signed** by default
 and the type **bit** is **unsigned** by default.

The **sign** property does not influence **real**, **shortreal**, and **string** types; effectively, **sign** can be ignored for these types.

E.3.4 Vector assignment

To create a packed array bit-string, the IP-XACT Expression Language allows the specification of vector information. Vector information can be specified only for parameters with a type of **bit**.

The following shows how to create a 8-bit packed array parameter with an identifier of **test** having the value 8'hAB:

```
<parameter type="bit" parameterId="test">
  <name>test</name>
  <vectors>
    <vector>
      <left>7</left><right>0</right>
    </vector>
  </vectors>
  <value>8'hAB</value>
</parameter>
```

which is the equivalent of

```
bit [7:0] test = 8'hAB;
```

E.3.5 Array assignment

To create an array for any type of parameter, the IP-XACT Expression Language allows the specification of array information and multiple initial values.

The following shows how to create an array parameter with a type of **int** and an identifier of **test** having the value '{ 8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA }:

```
<parameter type="int" parameterId="test">
  <name>test</name>
```

```

<arrays>
  <array>
    <left>6</left><right>0</right>
  </array>
</arrays>
<value>'{8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA}</value>
</parameter>

```

This is the equivalent of

```
longint test [0:6] = '{8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA};
```

The following shows how to create a packed bit array parameter with an identifier of `test` having the array value `{'{8'h0, 8'h1, 8'h2, 8'h3, 8'h4, 8'h5}, '{8'h6, 8'h7, 8'h8, 8'h9, 8'hA, 8'hB}}`:

```

<parameter type="bit" parameterId="test">
  <name>test</name>
  <vectors>
    <vector>
      <left>7</left><right>0</right>
    </vector>
  </vectors>
  <arrays>
    <array>
      <left>1</left><right>0</right>
      <left>5</left><right>0</right>
    </array>
  </arrays>
  <value>'{'{8'h0, 8'h1, 8'h2, 8'h3, 8'h4, 8'h5},
    '{8'h6, 8'h7, 8'h8, 8'h9, 8'hA, 8'hB}}</value>
</parameter>

```

This is the equivalent of

```
bit [0:7] test [0:1] [0:5] =
  {'{8'h0, 8'h1, 8'h2, 8'h3, 8'h4, 8'h5}, '{8'h6, 8'h7, 8'h8, 8'h9, 8'hA, 8'hB}};
```

E.3.6 Identifiers

The **parameterId** (see [C.6.2](#)), which is used as the identifier for the parameter by the expression language, can be specified using all characters from the *XML schema NameChar definition*; however, the first character of the **parameterId** shall not be a `-`, `:`, or `.`, a number, or any other character as specified by the *XML schema NameStartChar definition*.

The **parameterId** shall be unique within the containing document.

E.3.7 Identifier references

Parameters can be used in other expressions by referencing their identifier, the **parameterId**.

```

<ipxact:parameter parameterId="offset" type="int">
  <ipxact:name>offset</ipxact:name>
  <ipxact:value>'h100</ipxact:value>
</ipxact:parameter>
<ipxact:register>

```

```
...
<ipxact:addressOffset>offset + 'h10</ipxact:addressOffset>
...
</ipxact:register>
```

Any parameter in the document with a **parameterId** can be referenced from an expression. However, view-dependent parameters specified inside the instantiation sections cannot be referenced from outside the instantiation that specifies the parameter.

The **parameterId** can be specified using the characters `-`, `:`, or `..`. However, to avoid confusion with the expression operators, these characters cannot be directly specified when referencing the parameter from an expression. In that case, these characters should be escaped first (see [E.3.7.1](#)).

Example

```
shiftreg_a
busa_index
error_condition
merge_ab
_bus3
```

Implementations may set a limit on the maximum length of identifiers, but the limit shall be at least 1024 characters. If an identifier exceeds the implementation-specific length limit, an error shall be reported.

E.3.7.1 Escaped identifier references

Escaped identifier references shall start with the backslash character (`\`) and end with white space (a space, tab, or newline). They provide a means of including the characters that are part of the *XML schema NameChar definition*, but are not allowed to be specified directly for an identifier.

The following extra characters can be used for an escaped character: `-`, `:`, and `..`. Neither the leading backslash character nor the terminating white space is considered to be part of the identifier reference. Therefore, the escaped identifier reference `\cpu3` is treated the same as the non-escaped identifier references `cpu3`.

Example

```
\busa:index
\clock
\error-condition
\a.b
```

E.3.7.2 Keywords

Keywords are predefined non-escaped identifiers that are used to define the language constructs. All keywords are defined in lowercase only. A keyword preceded by an escape character (see [E.3.7.1](#)) is not interpreted as a keyword.

A subset of SystemVerilog keywords have been specified for IP-XACT. Since fewer keywords have been specified in the IP-XACT Expression Language, the other SystemVerilog keywords need to be escaped (see [E.3.7.1](#)) before processing these expressions using a SystemVerilog parser/processor.

The following keywords are reserved: **bit**, **byte**, **shortint**, **int**, **longint**, **shortreal**, **real**, **signed**, **unsigned**, **inside**, and **with**.

E.4 Operators

[Table E.2](#) gives an overview of the SystemVerilog operators and how IP-XACT supports them (or not).

Table E.2—SystemVerilog operators

Operator token	Name	Operand data types
=	Binary assignment operator	Assignments are not supported
+= -= /= *=	Binary arithmetic assignment operators	Assignments are not supported
%=	Binary arithmetic modulus assignment operator	Assignments are not supported
&= = ^=	Binary bit-wise assignment operators	Assignments are not supported
>>= <<=	Binary logical shift assignment operators	Assignments are not supported
>>>= <<<=	Binary arithmetic shift assignment operators	Assignments are not supported
?:	Conditional operator	integral ^a , real ^b , string
+ -	Unary arithmetic operators	integral ^a , real ^b
!	Unary logical negation operator	integral ^a , real ^b
~ & ~& ~ ^ ~^ ^~	Unary logical reduction operators	integral ^a
+ - * / **	Binary arithmetic operators	integral ^a , real ^b
%	Binary arithmetic modulus operator	integral ^a
& ^ ~^ ~^	Binary bit-wise operators	integral ^a
>> <<	Binary logical shift operators	integral ^a
>>> <<<	Binary arithmetic shift operators	integral ^a
&& -> <->	Binary logical operators	integral ^a , real ^b
< <= > >=	Binary relational operators	integral ^a , real ^b
==!=	Binary case equality operators	Unknown or high-impedance values are not supported
==!=	Binary logical equality operators	integral ^a , real ^b , string
==? !=?	Binary wildcard equality operators	Unknown or high-impedance values are not supported
++ --	Unary increment and decrement operators	Assignments are not supported
inside	Binary set membership operator	integral ^a , real ^b , string
dist	Binary distribution operator	Randomization is not supported
{ }{ }	Concatenation and replication operators	integral ^a , string
{<<} {>>{}}	Stream operators	integral ^a

^a integral = **bit**, **byte**, **shortint**, **int**, and **longint**

^b real = **real** and **shortreal**

All operators are supported except the following:

- a) The assignment operators are not supported, since assignments are handled using IP-XACT constructs.
- b) Operators that handle high-impedance and unknown equality are not supported. The IP-XACT Expression Language does not specify data-types that support high-impedance and unknown values; therefore, there is no need to support these operators.
- c) The binary distribution operator is not supported because it is not possible to specify randomized values.

E.5 Functions

The SystemVerilog Expression Language supports a set of functions; to make the expression language more complete, a set of functions is supported with additional functions specific to IP-XACT.

E.5.1 Integer function

The integer function is shown in [Table E.3](#).

Table E.3—Integer function

Function	Description
<code>\$clog2(x)</code>	Returns the ceiling of the log base 2 of the argument (the log rounded up to an integer value)

E.5.2 Real functions

The real functions are shown in [Table E.4](#).

Table E.4—Real functions

Function	Description
<code>\$ln(x)</code>	Natural logarithm
<code>\$log10(x)</code>	Decimal logarithm
<code>\$exp(x)</code>	Exponential
<code>\$sqrt(x)</code>	Square root
<code>\$pow(x,y)</code>	$x^{**}y$
<code>\$floor(x)</code>	Floor
<code>\$ceil(x)</code>	Ceiling
<code>\$sin(x)</code>	Sine
<code>\$cos(x)</code>	Cosine
<code>\$tan(x)</code>	Tangent

Table E.4—Real functions (continued)

\$asin(x)	Arc-sine
\$acos(x)	Arc-cosine
\$atan(x)	Arc-tangent
\$atan2(y,x)	Arc-tangent of y/x
\$hypot(x,y)	$\sqrt{x^2+y^2}$
\$sinh(x)	Hyperbolic sine
\$cosh(x)	Hyperbolic cosine
\$tanh(x)	Hyperbolic tangent
\$asinh(x)	Arc-hyperbolic sine
\$acosh(x)	Arc-hyperbolic cosine
\$atanh(x)	Arc-hyperbolic tangent

E.5.3 String function

The string function is shown in [Table E.5](#).

Table E.5—String function

Function	Description
\$sformatf()	Returns the IP-XACT string value specified by the initial format argument. Equivalent to the System Verilog \$sformatf() function, but, it operates on IP-XACT string values.

E.5.3.1 \$sformatf()

\$sformatf(format_string, [, list_of_arguments])

Returns IP-XACT string, or unresolvedStringExpression.

This is derived from the existing formatted string function from IEEE Std 1800-2012 System Verilog sections 21.3.3. The specification of format_string argument is derived from System Verilog section 21.2.1.2. Unlike System Verilog the format_string argument must be an IP-XACT expression with an IP-XACT string data type. The number of additional arguments must be at least the number of format escape sequences in the format string argument which require a value or there is an error. Additional arguments beyond the required number of arguments are formatted with “%d”. The data type of the additional arguments must match the data type allowed for the escape sequence or there is an error. For the argument type checking the argument type any of the unresolved data types is considered to be the same as the corresponding original data types. If one of the arguments other than the format_string is an unresolved data type then the return data type is unresolvedStringExpression. Otherwise, the return value is an IP-XACT string data type. The syntax of format string escape sequence is:

```
escape_sequence ::= basic_escape_sequence | real_escape_sequence
basic_escape_sequence ::= % [ width ] format
```

```

real_escape_sequence ::= % [width] . precision real_format
width ::= decimal_digit { decimal_digit }
precision ::= decimal_digit { decimal_digit }
format ::= basic_format | real_format
basic_format ::= B | C | D | H | O | S | X | b | c | d | h | o | s | x | %
real_format ::= E | F | G | e | f | g

```

If the escape sequence does not match the specified syntax the characters are placed into the result unchanged.

[Table E.6](#) shows a list of the IP-XACT format_string escape sequences:

Table E.6—Escape sequences

Escape Sequence	Description	IP-XACT Data Type Allowed
%h or %H %x or %X	Hexadecimal format. Lowercase format specifies lowercase result. Uppercase format specifies uppercase result.	bit, byte, shortint, int, longint; real and shortreal are first cast to longint
%d or %D	Decimal format.	bit, byte, shortint, int, longint; real and shortreal are first cast to longint
%o or %O	Octal format.	bit, byte, shortint, int, longint; real and shortreal are first cast to longint
%b or %B	Binary format.	bit, byte, shortint, int, longint; real and shortreal are first cast to longint
%c or %C	ACSII character format.	bit, byte, shortint, int, longint; real and shortreal are first cast to longint
%s or %S	String format.	string
%e or %E	Exponential format. Lowercase format specifies lowercase result. Uppercase format specifies uppercase result.	shortreal, real; bit, byte, shortint, int, longint are first cast to real
%f or %F	Decimal format	shortreal, real bit, byte, shortint, int, longint are first cast to real
%g or %G	Shorter of exponential or decimal. Lowercase format specifies lowercase result. Uppercase format specifies uppercase result.	shortreal, real bit, byte, shortint, int, longint are first cast to real
%%	The character %	No argument

The size of the resulting string is specified as in System Verilog section 21.2.1.3.

Example:

```
<command>$sformatf("run %c",byte unsigned'(A + 23))</command>
```

E.5.4 IP-XACT specific functions

The IP-XACT specific functions are shown in [Table E.7](#).

Table E.7—IP-XACT specific functions

Function	Description
<code>\$ipxact_absdefport_value(identifier)</code>	Returns an IP-XACT unresolved bit expression value representing an abstraction definition port value.
<code>\$ipxact_field_value (identifier)</code>	Returns an IP-XACT unresolved bit expression value representing an field slice value.
<code>\$ipxact_index_value (identifier)</code>	Returns an IP-XACT unresolved unsigned longint expression value representing the array index from the array dimension that has the indexVar attribute value matching the argument.
<code>\$ipxact_mode_condition (identifier)</code>	Returns an IP-XACT unresolved bit expression value representing a mode condition value.
<code>\$ipxact_packetfield_value (identifier)</code>	Returns an IP-XACT unresolved bit expression value representing a packet field value.
<code>\$ipxact_port_value (identifier)</code>	Returns an IP-XACT unresolved bit expression value representing a component port slice value.

E.5.4.1 \$ipxact_absdefport_value()

`$ipxact_absdefport_value (identifier)`

Returns IP-XACT unresolved bit expression.

Only allowed in the packetfield width value element.

The argument must be a reference to an abstraction definition port. It is an error if the abstraction definition port is not found.

Example:

```
<ipxact:width>$ipxact_absdefport_value(portRef1) == 4'ha</ipxact:width>
```

E.5.4.2 \$ipxact_field_value()

`$ipxact_field_value (field_slice_identifier)`

Returns IP-XACT unresolved bit expression.

Only allowed in the condition element value.

The argument must be a reference to a field slice. It is an error if the field slice is not found.

Example:

```
<ipxact:condition>$ipxact_field_value(fieldSliceRef1)==4'ha</
ipxact:condition>
```

E.5.4.3 \$ipxact_index_value()

\$ipxact_index_value (identifier)

Returns IP-XACT unresolved unsigned longint expression.

Only allowed in the pathSegment element value.

The argument must match the indexVar attribute of a dim element. It is an error if the indexVar is not found.

Example:

```
<ipxact:pathSegment>$sformatf("reg1[%0d]", $ipxact_index_value(N))</
ipxact:pathSegment>
```

E.5.4.4 \$ipxact_mode_condition()

\$ipxact_mode_condition (mode_identifier)

Returns IP-XACT unresolved bit expression.

Only allowed in the condition element value.

The argument must be a reference to a mode. It is an error if the mode is not found.

Example:

```
<ipxact:condition>$ipxact_mode_condition(mode1)</ipxact:condition>
```

E.5.4.5 \$ipxact_packetfield_value()

\$ipxact_packetfield_value (identifier)

Returns IP-XACT unresolved bit expression.

Only allowed in the packetfield width element.

The argument must be a reference to a packetfield. It is an error if the packetfield is not found.

Example:

```
<ipxact:width>$ipxact_packetfield_value(packetfieldRef1)</ipxact:width>
```

E.5.4.6 \$ipxact_port_value()

\$ipxact_port_value (identifier)

Returns IP-XACT unresolved bit expression.

Only allowed in the condition element value.

The argument must be a reference to a component port slice. It is an error if the port slice is not found. It is also an error if the port referenced in the referenced port slice is not found as a result of selecting a view in which that port does not exist. In other words, the conditions in which this function is called shall only be resolved in views in which the port slice exists.

Example:

```
<ipxact:condition>$ipxact_port_value(portSlice) == 4'ha</ipxact:condition>
```

E.5.5 IP-XACT specific escape sequences

The IP-XACT specific escape sequences are shown in [Table E.8](#).

Table E.8—IP-XACT escape sequences

Escape sequence	Description
\$ipxact_index_value(identifier)	IP-XACT DE or generator resolvable string value representing the array index from the array dimension that has the indexVar attribute value matching the argument.
\$ipxact_parameter_value(identifier)	IP-XACT DE or generator resolvable string value representing the parameter that has the parameterId attribute value matching the argument.

E.5.5.1 \$ipxact_index_value()

\$ipxact_index_value(identifier)

IP-XACT DE or generator resolvable string value.

Only allowed in displayName, description, and shortDescription values.

The argument must match the indexVar attribute of a dim element. It is an error if the indexVar is not found.

Example:

```
<ipxact:displayName>reg_$ipxact_index_value(N)</ipxact:displayName>
```

E.5.5.2 \$ipxact_parameter_value()

\$ipxact_parameter_value(identifier)

IP-XACT DE or generator resolvable string value.

Only allowed in displayName, description, and shortDescription values.

The argument must match the parameterId attribute of a parameter element. It is an error if the parameterId is not found.

Example:

```
<ipxact:displayName>component_$ipxact_parameter_value(P)</  
ipxact:displayName>
```

E.6 Expression language formal syntax (BNF)

The formal syntax is described using Backus-Naur Form (BNF). The syntax of the expression language source is derived from the starting symbol `constant_expression` (see [E.6.3.3](#)).

The following meta-syntaxis conventions used are used here:

- Keywords and punctuation are in **red** text.
- Syntactic categories are named in non-bold text.
- A vertical bar (|) separates alternatives.
- Square brackets ([]) enclose optional items.
- Braces ({ }) enclose items that can be repeated zero or more times.

E.6.1 Declarations: declaration data types

```
casting_type ::= simple_type | constant_primary | signing
```

```
integer_type ::= integer_vector_type | integer_atom_type
```

```
integer_atom_type ::= byte | shortint | int | longint
```

```
integer_vector_type ::= bit
```

```
non_integer_type ::= shortreal | real
```

```
signing ::= signed | unsigned
```

```
simple_type ::= integer_type | non_integer_type
```

E.6.2 Behavioral statements: Case statements

E.6.2.1 Patterns

```
assignment_pattern ::= '{ constant_expression { , constant_expression } }  
| '{ array_pattern_key : constant_expression { , array_pattern_key : constant_expression } }  
| '{ constant_expression { constant_expression { , constant_expression } } }
```

```
array_pattern_key ::= constant_expression | default
```

```
assignment_pattern_expression ::= [ assignment_pattern_expression_type ] assignment_pattern
```

```
assignment_pattern_expression_type ::= integer_atom_type
```

```
constant_assignment_pattern_expression ::= assignment_pattern_expression
```

E.6.2.2 Covergroup declarations

```
open_range_list ::= open_value_range { , open_value_range }
```

```
open_value_range ::= value_range
```

E.6.3 Expressions

E.6.3.1 Concatenations

```

constant_concatenation ::= { constant_expression { , constant_expression } }

constant_multiple_concatenation ::= { constant_expression constant_concatenation }

constant_streaming_expression ::= { stream_operator [ slice_size ] constant_stream_concatenation }

stream_operator ::= >> | <<

slice_size ::= simple_type | constant_expression

constant_stream_concatenation ::= { constant_stream_expression { , constant_stream_expression } }

constant_stream_expression ::= constant_expression [ with [ constant_array_range_expression ] ]

constant_array_range_expression ::= constant_expression
| constant_expression : constant_expression
| constant_expression +: constant_expression
| constant_expression -: constant_expression

```

E.6.3.2 Subroutine calls

```

math_function_call ::= math_function_identifier ( list_of_arguments )

string_function_call ::= string_function_identifier ( list_of_arguments )

ipxact_function_call ::= $ipxact_absdefport_value ( abstraction_definition_port_identifier )36
| $ipxact_field_value ( field_slice_identifier )37
| $ipxact_index_value ( indexvar_identifier )38
| $ipxact_mode_condition ( mode_identifier )39
| $ipxact_packetfield_value ( packetfield_identifier )40
| $ipxact_port_value ( port_slice_identifier )41

list_of_arguments ::= [ constant_expression ] { , [ constant_expression ] }

```

E.6.3.3 Expressions

```

constant_expression ::= constant_primary
| unary_operator constant_primary
| constant_expression binary_operator constant_expression
| constant_expression ? constant_expression : constant_expression
| constant_inside_expression

constant_range_expression ::= constant_expression | constant_part_select_range

constant_part_select_range ::= constant_range | constant_indexed_range

constant_range ::= constant_expression : constant_expression

constant_indexed_range ::= constant_expression +: constant_expression
| constant_expression -: constant_expression

constant_inside_expression ::= constant_expression inside { open_range_list }

value_range ::= constant_expression | [ constant_expression : constant_expression ]

```

³⁶The \$ipxact_absdefport_value() function only allowed in the packetfield width element value.

³⁷The \$ipxact_field_value() function only allowed in the condition element value.

³⁸The \$ipxact_index_value() function only allowed in the pathSegment element value.

³⁹The \$ipxact_mode_condition() function only allowed in the condition element value.

⁴⁰The \$ipxact_packetfield_value() function only allowed in the packetfield width element value.

⁴¹The \$ipxact_port_value() function only allowed in the condition element value.

E.6.3.4 Primaries

```

constant_primary ::= primary_literal
| parameter_identifier constant_select
| constant_concatenation [ [ constant_range_expression ] ]
| constant_multiple_concatenation [ [ constant_range_expression ] ]
| ( constant_expression )
| constant_streaming_expression
| constant_cast
| format_string_function_call
| math_function_call
| constant_assignment_pattern_expression
| condition_pattern42
| segment_pattern43
primary_literal ::= number | unbased_unsized_literal | string_literal

constant_cast ::= casting_type '( constant_expression )'

constant_bit_select ::= { [ constant_expression ] }

constant_select ::= constant_bit_select [ [ constant_part_select_range ] ]

```

E.6.3.5 Operators

```

unary_operator ::= + | - | ! | ~ | & | ~& | || | ~| | ^ | ~^ | ^~
binary_operator ::= + | - | * | / | % | == | != | && | || | ** | < | <= | > | >= | & | || | ^ | ~ | ~^ | >> | << | >>> | <<< | -> | <>

```

E.6.3.6 Numbers

See A.8.7 in IEEE Std 1800-2012.

E.6.3.7 Strings

```
string_literal ::= " { Any_ASCII_Characters } "
```

E.6.4 General: Identifiers

```

abstraction_definition_port_identifier ::= identifier

escaped_identifier ::= \ {any_printable_ASCII_character_except_white_space} white_space

field_slice_identifier ::= identifier

identifier ::= simple_identifier | escaped_identifier

indexvar_identifier ::= identifier

integer_math_function_identifier ::= $clog2

math_function_identifier ::= integer_math_function_identifier | real_math_function_identifier

mode_identifier ::= identifier

packetfield_identifier ::= identifier

parameter_identifier ::= identifier

port_slice_identifier ::= identifier

```

⁴² Alternative only valid on unresolve type of kind mode/condition expression.

⁴³ Alternative only valid on unresolve type of kind segment_path expression.

```
simple_identifier ::= [ a-zA-Z_ ] { [ a-zA-Z0-9_ $ ] }  
real_math_function_identifier ::= $asin | $ln | $acos | $log10 | $atan | $exp | $atan2 | $sqrt | $hypot | $pow |  
$sinh | $floor | $cosh | $ceil | $tanh | $sin | $asinh | $cos | $acosh | $tan | $satanh  
string_function_identifier ::= $sformatf
```

E.7 SystemVerilog conversion steps

To be able to verify the IP-XACT Expression Language using a SystemVerilog parser or tool, the IP-XACT Expression Language needs to be converted to the SystemVerilog Expression language.

This can be achieved using the following (simple) steps.

E.7.1 Convert parameter

- a) Convert the vector(s) left and right information to SystemVerilog (packed) array indexes.
- b) Normalize the characters in the name of the parameter to valid SystemVerilog characters by replacing all -, :, and . characters and non US-ASCII characters with a predefined character (or set of characters), e.g., use an underscore (_) instead.
- c) Escape all identifiers that could be interpreted as SystemVerilog keywords.
- d) Convert the array(s) left and right information to SystemVerilog (unpacked) array indexes.
- e) Convert the value and array values (see [E.7.2](#)).

E.7.2 Convert expression

Convert all identifiers by normalizing the characters: use the same mechanism as specified by steps [c](#)) and [d](#)) in [E.7.1](#).

E.8 SystemVerilog reference

The following subclauses of IEEE Std 1800-2012 have been used to determine the IP-XACT Expression Language functionality:

6 – Data Types: the following subclauses should be considered:

- **6.1 - General:** only the parts discussing the type operators, compatibility, and casting.
- **6.2 - Data types and data objects:** should be considered, apart from any distinctions noted in [E.2](#).
- **6.3 - Value Set:** only the subclause about the logical 0 and 1 values ([6.3.1](#)); the subclauses about strength should not be taken into consideration.
- **6.4 - Singular and aggregate types, 6.5 - Nets and variables, 6.6 - Net types, 6.7 - Net declarations, and 6.8 - Variable declarations:** should not be considered.
- **6.9 - Vector declarations:** only bit vectors should be considered. Also the declaration of vectors is handled by an IP-XACT construct and can be ignored.
- **6.10 - Implicit declarations:** can be ignored because identifiers can be created only using the IP-XACT parameter mechanism
- **6.11 - Integer data types:** only **byte**, **shortint**, **int**, **longint**, and **bit** should be considered; these are 2-state data-types; therefore, all 4-state data types can be ignored.

- **6.12 - Real, shortreal and realtime data types:** only the real data-type should be considered.
- **6.13 - Void data type, 6.14 - Chandle data type, and 6.15 - Class:** are not supported by the IP-XACT Expression Language and should not be considered.
- **6.16 - String data type:** the IP-XACT string data-type functionality has been simplified considerably; only the equality, inequality, concatenation and replication operators should be considered.
- **6.17 - Event data type, 6.18 - User-defined types, 6.19 - Enumerations, and 6.20 - Constants:** cannot be specified using the IP-XACT expression language and should not be considered.
- **6.21 - Scope and lifetime:** is handled by the *IP-XACT XML Format*, which is outside of the IP-XACT Expression Language.
- **6.22 - Type compatibility:** only the parts that discuss the **byte**, **shortint**, **int**, **longint**, **bit**, **shortreal**, and **real** data-types should be considered.
- **6.23 - Type operator:** the type operator is not supported and should not be considered.
- **6.24 - Casting:** only the parts that discuss the **byte**, **shortint**, **int**, **longint**, **bit**, **shortreal**, and **real** data-types should be considered.
- **6.25 - Parameterized data types:** should not be considered.

7 – Aggregate data types: only the parts about packed arrays and unpacked arrays should be considered.

- **7.1 - General:** only the part on packed and unpacked arrays should be considered.
- **7.2 - Structures and 7.3 – Unions:** are not supported and should not be considered.
- **7.4 - Packed and unpacked arrays:** should be fully considered.
- **7.5 - Dynamic arrays:** are not supported and should not be considered.
- **7.6 - Array assignments:** are handled by the *IP-XACT XML Format*; however, the format of the assignment should still be considered.
- **7.7 - Arrays as arguments to subroutines:** subroutines are not supported and should not be considered.
- **7.8 - Associative arrays:** are not supported and should not be considered.
- **7.9 - Associative array methods:** are not supported and should not be considered.
- **7.10 - Queues:** are not supported and should not be considered.
- **7.11 - Array querying functions and 7.12 - Array manipulation methods:** are not supported and should not be considered.

10 – Assignment statements: only **10.9 - Assignment patterns** should be considered, in particular **10.9.1**, which clarifies the array assignment patterns.

11 – Operators and expressions: the following subclauses should be considered:

- **11.1 - General:** should be considered, apart from the operators and expressions discussed in [E.4](#).
- **11.2 - Overview:** where for the operand subclause only, constant literal numbers, string literals, packed array, and unpacked arrays should be considered.
- **11.3 - Operators:** only the operators identified for the IP-XACT Expression Language should be considered and only on the data-types allowed. The assignment within an expression section can be ignored.
- **11.4 - Operator descriptions:** only Arithmetic operators, Relational operators, Equality operators (x and z excluded), Logical operators, Bitwise operators, Reduction operators, Shift operators, Conditional operators, Concatenation, Set membership, and streaming operators should be considered.
- **11.5 - Operands:** should be considered.

- **11.6 - Expression bit lengths:** should be considered for the **byte**, **shortint**, **int**, **longint**, **bit**, **short-real**, and **real** values.
- **11.7 - Signed expressions:** the **\$signed** and **\$unsigned** system functions are not supported and should be ignored.
- **11.8 - Expression evaluation rules:** should be considered, however the subclauses about evaluating an assignment and handling of X and Z values can be ignored.
- **11.9 - Tagged union expressions and member access:** unions are not supported and can be ignored.
- **11.10 - String literal expressions:** are not supported and can be ignored.
- **11.11 - Operator overloading:** is not supported and can be ignored.
- **11.12 - Minimum, typical, and maximum delay expressions:** delay expressions are not supported and can be ignored.
- **11.13 - Let construct:** this construct is not supported and can be ignored.

20 – Utility system tasks and system functions: only **20.8 – Math functions** should be considered.

Annex F

(normative)

Tight generator interface

IP-XACT generators are tools that are invoked from within a DE to perform an operation required by the user of the DE. For example, generators can be provided to verify the configuration of a subsystem, generate an address map, or write a netlist representation of the subsystem in a target language such as Verilog or SystemC. To perform their various operations, most generators need access to the IP-XACT meta-data describing the subsystem, as currently loaded into the DE. Generators need both read- and write-access to the IP-XACT meta-data. All generators are external applications running in a separate address space from the DE.

The TGI defines how the DE and generator cooperate to achieve the desired end-goal of the user of the DE. The TGI defines the method of communication between the DE and generator, the method for invoking the generator, and the actual application programming interface (API) that can be used to read and write the IP-XACT meta-data stored in the DE. [F.1](#), [F.2](#), and [F.3](#) describe each of these three aspects of the TGI, respectively.

F.1 Method of communication

The DE and the generator communicate with each other by sending messages to each other utilizing the SOAP standard. SOAP or REST provides a simple means for sending XML-format messages using HTTP or other transport protocols. The TGI restricts the set of allowed transport protocols to HTTP and a file-based protocol. All generators are required to support HTTP, but support for the file-based protocol is optional. The same rules apply to the DE—it shall support the use of HTTP, but is not required to support the file-based protocol, even though a generator may allow it. The protocols supported by a generator are specified using the **transportMethod** element within the **componentGenerator** element.

The information required to use a particular transport protocol shall be passed to the generator by the DE when it is invoked, as described in [F.2](#). For HTTP, the generator is passed a URL of the form **http://host_name:port_number**. All SOAP or REST messages sent to the DE shall be sent using the referenced URL. For the file-based protocol, the generator is passed a URL of the form **file://file_name**. In this case, all SOAP or REST messages are written to the specified file.

Each DE and generator is responsible for setting itself up to communicate using SOAP or REST with the appropriate transport protocol. For example, a generator written in Tcl might include the Tcl SOAP package to enable SOAP functionality. Once the communication channel is set up, the generator can read and write the IP-XACT meta-data using any legal SOAP message. The set of legal SOAP and REST messages defines the API portion of the TGI (see [F.7](#)).

F.2 Generator invocation

All of the information known by the DE about a particular generator comes from an instance of the **componentGenerator** (see [6.16](#)), **abstractorGenerator** (see [8.9](#)), or **generator** (see [10.4](#)) elements. These elements provides the following information:

- a) **name** is the name of the generator as seen within the DE.
- b) **executable** is the URL defining the location of the generator.

- c) **parameters** is a list of name/value pairs defining information to be passed to the generator.
- d) **apiType** indicates the generator type: **TGI** or **none** (no communication).
- e) **transportMethods** show any transport mechanisms supported (in addition to HTTP).
- f) **phase** (not relevant to the TGI).
- g) **vendorExtensions** (not relevant to the TGI).
- h) **group** (not relevant to the TGI).

F.2.1 Resolving the URL

The URL defining the generator executable shall resolve to one of the following forms:

- **file:path_to_executable** (e.g., `file:/usr/jdoe/bin/mygen.pl` or `file:../bin/mygen.pl`) defines the path for invoking the generator on the machine from which the DE was invoked.
- **file://machine_name/path_to_executable** (e.g., `file://server1/tmp/othergen.pl`) defines the path for invoking a generator on the specified machine.
- **http://web_address:port_number** (e.g., `http://www.acme.com/generator:1500`) defines the URL of a generator implemented as a Web-based server.

All *file references* are relative to the location of the XML description in which the file reference is contained.

For the file-based generators, the DE shall invoke the generator as a sub-process with a command line built up as:

executable -url transport_URL generator_parameter_arguments

The *generator_parameter_arguments* are the parameters from the **componentGenerator** element with the user-specified values. Each parameter causes two additional arguments to be passed to the generator with the following format: *-parameter_name parameter_value*. The DE needs to create a *transport_URL* that specifies a protocol supported by the generator as defined by the transport methods within the **componentGenerator**. The DE is also responsible for ensuring any passed parameters can be interpreted correctly. This URL is to be used in the generator to set up the SOAP or REST communication channel.

For Web-based generators, the DE shall send a message to the address and port defined as the executable. The format of this message is

url=transport_URL&generator_parameter_arguments

In this case, the generator parameters are formatted using the standard HTTP parameter passing syntax. The specified transport URL shall be used by the generator for any return messages to the DE.

The invocation syntax described above applies only to generators with an API type of **TGI**. Generators with an API type of **none** are invoked as described above, excluding the **transport_URL** argument.

F.2.2 Example

This example shows file-based and Web-based **componentGenerator** elements.

```
<ipxact:componentGenerator>
```

```

<ipxact:name>myGenerator</ipxact:name>
<ipxact:parameters>
    <ipxact:parameter resolve="user" parameterId="param1_id">
        <ipxact:name>param1</ipxact:name>
        <ipxact:value>"default1"</ipxact:value>
    </ipxact:parameter>
    <ipxact:parameter>
        <ipxact:name>param2</ipxact:name>
        <ipxact:value>"fixedValue"</ipxact:value>
    </ipxact:parameter>
</ipxact:parameters>
<ipxact:apiType>TGI_2022_BASE</ipxact:apiType>
<ipxact:transportMethods>
    <ipxact:transportMethod>file</ipxact:transportMethod>
</ipxact:transportMethods>
<ipxact:generatorExe>../bin/myGenerator.pl</ipxact:generatorExe>
</ipxact:componentGenerator>

```

produces the following output:

```

path_to_XML/..../bin/myGenerator -url http://host:port -param1 default1
-param2 fixedValue

```

Whereas:

```

<ipxact:componentGenerator>
    <ipxact:name>myWebGenerator</ipxact:name>
    <ipxact:parameters>
        <ipxact:parameter resolve="user" parameterId="param1_id">
            <ipxact:name>param1</ipxact:name>
            <ipxact:value>"default1"</ipxact:value>
        </ipxact:parameter>
        <ipxact:parameter>
            <ipxact:name>param2</ipxact:name>
            <ipxact:value>"fixedValue"</ipxact:value>
        </ipxact:parameter>
    </ipxact:parameters>
    <ipxact:apiType>TGI_2022_BASE</ipxact:apiType>
    <ipxact:generatorExe>http://www.acme.com:1500</ipxact:generatorExe>
</ipxact:componentGenerator>

```

produces the following output:

```

http://www.acme.com:1500?url=http%3a%2f%2fhost%3aport&param1=default1
&param2=fixedValue

```

F.3 TGI API

The TGI API defines the set of legal SOAP or REST messages that can be sent from a generator to a DE, along with the format of the responses the generator can expect from a given request (message) to the DE.

The API shall provide the means of getting and setting values within the IP-XACT design currently represented in the DE. The API commands can be classified as shown in [Table F1](#).

Table F1—TGI API classifications

Classification	Description	Example
Administrative	Commands that do not deal directly with the IP-XACT meta-data.	Terminate communication.
Traversal	Commands that return a list of elements, which can then be traversed for further manipulation.	Get components in a design.
Create	Commands that create new top elements. These commands are only available in the TGI Extended mode.	Create a new component.
Add	Commands that add a child element to a parent element. These commands are only available in the TGI Extended mode.	Add a busInterface to a component.
Remove	Commands that remove an element from its parent element. These commands are only available in the TGI Extended mode.	Remove a busInterface from a component.
Get	Commands that get attribute or element values. These commands are available for getting all information from the design and component schemas. If the attribute or element does not exist, this may return a default value, an empty string, or an empty array.	Get port width.
Set	Commands that set element value or expression. If the element is not present, the set may create the element and assign it the given value. Set routines return a Boolean value where a true return code implies a successful operation. If false is returned, the SOAP or REST fault code shall provide additional information detailing the failure.	Set parameter value.

The complete set of API commands is defined using WSDL for the SOAP protocol or YAML for the REST protocol, so that it can be defined in a language-independent format.

F.3.1 TGI fault codes

The fault codes for TGI failures are as follows:

- 1 - Unknown (undefined) error
- 2 - Illegal element ID
- 3 - Illegal value(s)
- 4 - Element is not modifiable (incompatible **resolve** value)
- 5 - Operation not supported by the DE
- 6 - Operation not supported in this version of the schema
- 7 - Operation failed

F.3.2 Administrative commands

There are three administrative commands defined in the API.

- a) **Init** is the required first message from the generator to the DE. It tells the DE that the generator has properly connected via the specified communication protocol (SOAP or REST).
 - 1) Input
 - i) **apiVersion** of type *string*—Indicates the API version with which the generator is defined to work.
 - ii) **failureMode** of type *apiFailureMode*—Compatibility failure mode
 - fail** indicates the DE shall return an error on the *init* call if its API version does not match the one passed to the *init* call;
 - error** indicates the DE shall return an error each time a potentially incompatible API call is made;
 - warning** indicates the DE shall increment a warning count each time a potentially incompatible API call is made.
 - iii) **message** of type *string*—Message that the DE may display to the user.
 - 4) Returns: **status** of type *boolean*.
- b) **End** is the required last message from the generator to the DE. It tells the DE it is okay to stop listening for messages from the generator. This includes a generator return status, although the generator is not strictly required to terminate after sending the message.
- c) **Message** indicates some form of generator status to pass to the user.

F.3.3 Return values

The TGI commands can return values of the following types:

- Boolean
- String
- List of Strings
- Long
- Double
- Float

Each basic type includes value NULL, which may be returned in case a value that is asked for is not defined.

F.4 IDs and configurable values

The *handles* in TGI are classified as **instanceIDs** and **IDs**. The **instanceIDs** reference configured entities while the **IDs** reference unconfigured entities. The following TGI calls show the difference between **instanceID** and **ID** explicitly for top-level elements and configurable elements:

instanceID = **abstractionDefInstanceID** | **busDefInstanceID** | **componentInstanceID** | **abstractorInstanceID** | **designInstanceID** | **designConfigurationInstanceID** | **typeDefinitionsInstanceID**

ID = **abstractionDefID** | **busDefID** | **componentID** | **abstractorID** | **designID** | **designConfigurationID** | **generatorChainID** | **typeDefinitionsID**

For each **instanceID**, the **getUnconfiguredID(instanceID)** can be called to retrieve the **ID**. For the calls that retrieve information (get calls), it is presumed the call on an **instanceID** also returns **instanceIDs**, e.g., **getComponentBusInterfaceIDs(componentInstanceID)** returns **busInterfaceInstanceIDs** (configured), whereas **getComponentBusInterfaceIDs(componentID)** returns **busInterfaceIDs** (unconfigured). For clarity, the distinction between **instanceIDs** and **IDs** has not been made for non-top-level elements. Calls that modify information (edit or set calls), operate only on **IDs** (unconfigured), e.g.,

addComponentInitiatorBusInterface(*componentID*) returns a new unconfigured **busInterfaceID**, whereas the call **addComponentInitiatorBusInterface(*componentInstanceID*)** fails.

Handles returned by TGI commands are persistent for the duration of a single generator invocation provided the element being referenced is not removed. For example, if a handle represents an address space element, that handle can be utilized as often as is needed during a single generator invocation, unless the component containing the address map is removed via **removeDesignComponentInstance()**. Furthermore, persistent TGI handles to the same object are identical for the duration of the generator invocation. This enables generators to identify objects by means of their handles.

F.5 TGI messages

The TGI is a set of messages used to query and modify an IP-XACT-compliant database. For the SOAP protocol, the TGI messages are composed of an envelope and a TGI body. The TGI services are specified in the **TGI.wsdl** file. Each TGI body message is an XML element whose name is the name of the TGI command and whose elements are the arguments of the TGI command. All TGI messages apply to IP-XACT XML elements, identified by an ID, i.e., a TGI server-defined constant uniquely identifying an IP-XACT XML element throughout a TGI server session.

F.6 Vendor attributes

One case of special interest to a user may be the location of vendor attributes in the schema. These attributes are allowed in more places in the schema than the TGI allows a user to retrieve them; this goes back to the concept where one function uses many different ID types to return some data. Regardless, vendor attributes can be accessed only if the containing element has an ID.

F.7 TGI calls

This subclause details the TGI API calls. [F.7.1](#) is an index of the various categories, and [F.7.2](#) is an index for the specific messages within each of those categories. The actual API breakouts start with subclause [F.7.3](#).

F.7.1 Category index

Base category name	Extended category name
Abstraction definition (BASE)	Abstraction definition (EXTENDED)
Abstractor (BASE)	Abstractor (EXTENDED)
Access Policy (BASE)	
Access handle (BASE)	Access handle (EXTENDED)
Access policy (BASE)	Access policy (EXTENDED)
Address space (BASE)	Address space (EXTENDED)
Array (BASE)	Array (EXTENDED)

Assertion (BASE)	Assertion (EXTENDED)
Bus definition (BASE)	Bus definition (EXTENDED)
Bus interface (BASE)	Bus interface (EXTENDED)
CPU (BASE)	CPU (EXTENDED)
Catalog (BASE)	Catalog (EXTENDED)
Choice (BASE)	Choice (EXTENDED)
Clearbox (BASE)	Clearbox (EXTENDED)
Component (BASE)	Component (EXTENDED)
Configurable element (BASE)	Configurable element (EXTENDED)
Constraint (BASE)	Constraint (EXTENDED)
Constraint Set (BASE)	Constraint Set (EXTENDED)
Design (BASE)	Design (EXTENDED)
Design configuration (BASE)	Design configuration (EXTENDED)
Driver (BASE)	Driver (EXTENDED)
Element attribute (BASE)	Element attribute (EXTENDED)
File builder (BASE)	File builder (EXTENDED)
File set (BASE)	File set (EXTENDED)
Generator (BASE)	Generator (EXTENDED)
Generator chain (BASE)	Generator chain (EXTENDED)
Indirect interface (BASE)	Indirect interface (EXTENDED)
Instantiation (BASE)	Instantiation (EXTENDED)
Memory map (BASE)	Memory map (EXTENDED)
Miscellaneous (BASE)	Miscellaneous (EXTENDED)
Module parameter (BASE)	Module parameter (EXTENDED)
Name group (BASE)	Name group (EXTENDED)
Parameter (BASE)	Parameter (EXTENDED)
Port (BASE)	Port (EXTENDED)
Port map (BASE)	Port map (EXTENDED)
Power (BASE)	Power (EXTENDED)

Register (BASE)	Register (EXTENDED)
Register file (BASE)	Register file (EXTENDED)
Slice (BASE)	Slice (EXTENDED)
Top element (BASE)	Top element (EXTENDED)
Type definitions (BASE)	Type definitions (EXTENDED)
Vector (BASE)	Vector (EXTENDED)
Vendor extensions (BASE)	Vendor extensions (EXTENDED)
View (BASE)	View (EXTENDED)

F.7.2 Abstraction definition (BASE)

F.7.2.1 getAbstractionDefBusTypeRefByVNV

Description: Returns the busType VNV defined on the given abstractionDef or abstractionDefInstance element.

- Returns: VNV of type **String** - The VNV of the referenced busDefinition object
- Input: absDefOrAbsDefInstanceID of type **String** - Handle to an abstractionDef or abstractionDefInstance element

F.7.2.2 getAbstractionDefChoiceIDs

Description: Returns the handles to all the choices defined on the given abstraction definition object element.

- Returns: choiceIDs of type **String** - List of handles to choice elements
- Input: abstractionDefinitionID of type **String** - Handle to an abstraction definition object element

F.7.2.3 getAbstractionDefExtendsRefByVNV

Description: Returns the extended VNV defined on the given abstractionDefinition object.

- Returns: VNV of type **String** - The VNV of the extended abstractionDefinition object
- Input: abstractionDefinitionID of type **String** - Handle to an abstractionDefinition object

F.7.2.4 getAbstractionDefPortIDs

Description: Returns the handles to all the ports defined on the given abstractionDefinition object.

- Returns: portIDs of type **String** - List of handles to the ports elements
- Input: abstractionDefinitionID of type **String** - Handle to an abstractionDefinition object

F.7.2.5 getAbstractionDefPortLogicalName

Description: Returns the logicalName element defined on the given abstractionDefPort element.

- Returns: logicalName of type **String** - The logical port name
- Input: abstractionDefPortID of type **String** - Handle to an abstractionDefPort element

F.7.2.6 getAbstractionDefPortMatch

Description: Returns the match of the given abstractionDefPort element.

- Returns: `match` of type ***Boolean*** - The logical port match
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.2.7 getAbstractionDefPortOnSystemIDs

Description: Returns the handles to all the onSystem elements defined on the given abstractionDefPort element.

- Returns: `onSystemIDs` of type ***String*** - List of handles to the onSystem elements
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.2.8 getAbstractionDefPortPacketIDs

Description: Returns the handles to all the portPackets defined on the given abstractionDef port element.

- Returns: `portPacketIDs` of type ***String*** - List of portPacketType handles
- Input: `abstractionDefPortID` of type ***String*** - Handle to an abstractionDefinition port element

F.7.2.9 getAbstractionDefPortStyle

Description: Returns the port mode element defined on the given abstractionDefPort element.

- Returns: `mode` of type ***String*** - The logical port mode (onInitiator, onTarget, onSystem)
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to an abstractionDefPort element

F.7.2.10 getAbstractionDefPortTransactionalModeBusWidth

Description: Returns the busWidth value defined on the given abstractionDef port element.

- Returns: `busWidth` of type ***Long*** - The port busWidth value
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to a logical port transactional element

F.7.2.11 getAbstractionDefPortTransactionalModeBusWidthExpression

Description: Returns the busWidth expression defined on the given abstractionDef port element.

- Returns: `busWidth` of type ***String*** - The port busWidth expression
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to a logical port transactional element

F.7.2.12 getAbstractionDefPortTransactionalModeBusWidthID

Description: Returns the handle to the busWidth defined on the given abstractionDef port element.

- Returns: `busWidthID` of type ***String*** - Handle to the busWidth element
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to a logical port transactional element

F.7.2.13 getAbstractionDefPortTransactionalModeInitiative

Description: Returns the value of the initiative element defined on the given abstractionDef port element.

- Returns: initiative of type **String** - The port initiative
- Input: abstractionDefPortModeID of type **String** - Handle to a logical port transactional element

F.7.2.14 getAbstractionDefPortTransactionalModeKindID

Description: Returns the handle to the kind defined on the given abstractionDef port element.

- Returns: kindID of type **String** - Handle to the kind element
- Input: abstractionDefPortModeID of type **String** - Handle to a logical port transactional element

F.7.2.15 getAbstractionDefPortTransactionalModePresence

Description: Returns the value of the presence element defined on the given abstractionDef port element.

- Returns: presence of type **String** - The port presence
- Input: abstractionDefPortModeID of type **String** - Handle to a logical port transactional element

F.7.2.16 getAbstractionDefPortTransactionalModeProtocolID

Description: Returns the handle to the protocol defined on the given abstractionDef port element.

- Returns: protocolID of type **String** - Handle to the protocol element
- Input: abstractionDefPortModeID of type **String** - Handle to a logical port transactional element

F.7.2.17 getAbstractionDefPortTransactionalOnInitiatorID

Description: Returns the handle to the onInitiator element defined on the given abstractionDefinition port transactional element.

- Returns: onInitiatorID of type **String** - Handle to the onInitiator element
- Input: abstractionDefPortID of type **String** - Handle to a transactional port element

F.7.2.18 getAbstractionDefPortTransactionalOnSystemIDs

Description: Returns the handles to all the onSystem elements defined on the given abstractionDef port transactional element.

- Returns: onSystemIDs of type **String** - List of handles to the onSystem elements
- Input: abstractionDefPortID of type **String** - Handle to a logical port transactional element

F.7.2.19 getAbstractionDefPortTransactionalOnTargetID

Description: Returns the handle to the onTarget element defined on the given abstractionDefinition port transactional element.

- Returns: onTargetID of type **String** - Handle to the onTarget element
- Input: abstractionDefPortID of type **String** - Handle to a transactional port element

F.7.2.20 getAbstractionDefPortTransactionalQualifierID

Description: Returns the handle to the qualifier defined on the given abstractionDef port transactional element.

- Returns: `qualifierID` of type ***String*** - Handle to the qualifier element
- Input: `portID` of type ***String*** - Handle to a port element

F.7.2.21 getAbstractionDefPortWireDefaultValue

Description: Returns the defaultValue defined on the given abstractionDef logical port element.

- Returns: `value` of type ***Long*** - The logical port default value
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.2.22 getAbstractionDefPortWireDefaultValueExpression

Description: Returns the defaultValue expression defined on the given abstractionDef logical port element.

- Returns: `expression` of type ***String*** - The logical port defaultValue expression
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.2.23 getAbstractionDefPortWireDefaultValueID

Description: Returns the handle to the defaultValue defined on the given abstractionDef port element.

- Returns: `defaultValueID` of type ***String*** - Handle to the defaultValue element
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to a logical port wire element

F.7.2.24 getAbstractionDefPortWireModeDirection

Description: Returns the direction defined on the given abstractionDef port element.

- Returns: `direction` of type ***String*** - The port direction
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to a logical port wire element

F.7.2.25 getAbstractionDefPortWireModeMirroredModeConstraintsID

Description: Returns the handle to the mirroredModeConstraints defined on the given abstractionDef port element.

- Returns: `mirroredModeConstraintsID` of type ***String*** - Handle to the mirroredModeConstraints element
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to a logical port wire element

F.7.2.26 getAbstractionDefPortWireModeModeConstraintsID

Description: Returns the handle to the modeConstraints defined on the given abstractionDef port element.

- Returns: `modeConstraintsID` of type ***String*** - Handle to the modeConstraints element
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to a logical port wire element

F.7.2.27 getAbstractionDefPortWireModePresence

Description: Returns the presence defined on the given abstractionDef port element.

- Returns: `presence` of type ***String*** - The presence value
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to a logical port wire element

F.7.2.28 getAbstractionDefPortWireModeWidth

Description: Returns the width value defined on the given abstractionDef port element.

- Returns: width of type **Long** - The width value
- Input: portModeID of type **String** - Handle to a logical port wire element

F.7.2.29 getAbstractionDefPortWireModeWidthExpression

Description: Returns the width expression defined on the given abstractionDef port element.

- Returns: width of type **String** - The width expression
- Input: portModeID of type **String** - Handle to a logical port wire element

F.7.2.30 getAbstractionDefPortWireModeWidthID

Description: Returns the handle to the width defined on the given abstractionDef port element.

- Returns: widthID of type **String** - Handle to the width element
- Input: portModeID of type **String** - Handle to a logical port wire element

F.7.2.31 getAbstractionDefPortWireOnInitiatorID

Description: Returns the handle to the onInitiator element defined on the given abstractionDef port element.

- Returns: onInitiatorID of type **String** - Handle to the onInitiator element
- Input: abstractionDefPortModeID of type **String** - Handle to a logical port wire element

F.7.2.32 getAbstractionDefPortWireOnSystemIDs

Description: Returns the handles to all the onSystem elements defined on the given abstractionDef port wire element.

- Returns: onSystemIDs of type **String** - List of handles to the onSystem elements
- Input: portID of type **String** - Handle to a logical port wire element

F.7.2.33 getAbstractionDefPortWireOnTargetID

Description: Returns the handle to the onTarget defined on the given abstractionDef port.

- Returns: onTargetID of type **String** - Handle to the onTarget element
- Input: abstractionDefPortID of type **String** - Handle to a logical port wire element

F.7.2.34 getAbstractionDefPortWireQualifierID

Description: Returns the handle to the qualifier defined on the given abstractionDef port wire element.

- Returns: qualifierID of type **String** - Handle to the qualifier element
- Input: portID of type **String** - Handle to a port element

F.7.2.35 getAbstractionDefPortWireRequiresDriver

Description: Returns the requiresDriver element defined on the given abstractionDefPort element.

- Returns: value of type **Boolean** - The logical port requiresDriver value
- Input: abstractionDefPortID of type **String** - Handle to an abstractionDefPort element

F.7.2.36 getAbstractionDefPortWireRequiresDriverID

Description: Returns the handle to the requiresDriver defined on the given abstractionDefPort element.

- Returns: requiresDriverID of type **String** - Handle to the requiresDriver element

- Input: abstractionDefPortModeID of type **String** - Handle to a logical port wire element

F.7.2.37 getModeConstraintsDriveConstraintCellSpecificationID

Description: Returns the handle to the driveConstraint defined on the given modeConstraints element.

- Returns: driveConstraintID of type **String** - Handle to the driveConstraint element
- Input: modeConstraintsID of type **String** - Handle to a modeConstraints element

F.7.2.38 getModeConstraintsLoadConstraintID

Description: Returns the handle to the loadConstraint defined on the given modeConstraints element.

- Returns: loadConstraintID of type **String** - Handle to the loadConstraint element
- Input: modeConstraintsID of type **String** - Handle to a modeConstraints element

F.7.2.39 getModeConstraintsTimingConstraintIDs

Description: Returns the handles to all the timingConstraints defined on the given modeConstraints element.

- Returns: timingConstraintIDs of type **String** - List of handles to the timingConstraint elements
- Input: modeConstraintsID of type **String** - Handle to a modeConstraints element

F.7.2.40 getOnSystemGroup

Description: Returns the group attribute defined the given onSystem element.

- Returns: group of type **String** - The group name
- Input: onSystemID of type **String** - Handle to an onSystem element

F.7.2.41 getPacketEndianness

Description: Returns the endianness defined on the given packet element.

- Returns: endianness of type **String** - The endianness (big or little)
- Input: packetID of type **String** - Handle to a packet element

F.7.2.42 getPacketFieldEndianness

Description: Returns the endianness defined on the given packetField element.

- Returns: endianness of type **String** - The endianness (big or little)
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.2.43 getPacketFieldQualifierID

Description: Returns the handle to the qualifier defined on the given packetField element.

- Returns: qualifierID of type **String** - Handle to the qualifier element
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.2.44 getPacketFieldValue

Description: Returns the (resolved) value of the value element defined on the given packetField element.

- Returns: value of type **Long** - The value of the value element
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.2.45 getPacketFieldValueExpression

Description: Returns the expression defined on the value element of the given packetField element.

- Returns: valueExpression of type **String** - The expression defined on the value element
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.2.46 getPacketFieldValueID

Description: Returns the handle to the value element defined on the given packetField element.

- Returns: valueID of type **String** - Handle to the value element
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.2.47 getPacketFieldWidth

Description: Returns the (resolved) width value defined on the given packetField element.

- Returns: width of type **Long** - The value of the width element
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.2.48 getPacketFieldWidthExpression

Description: Returns the width expression defined on the given packetField element.

- Returns: width of type **String** - The width expression
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.2.49 getPacketFieldWidthID

Description: Returns the handle to the width element defined on the given packetField element.

- Returns: widthID of type **String** - Handle to the width element
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.2.50 getPacketPacketFieldIDs

Description: Returns the handles to all the packetFields defined on the given packet element.

- Returns: packetFieldIDs of type **String** - List of handles to packetField elements
- Input: packetID of type **String** - Handle to a packet element

F.7.3 Abstraction definition (EXTENDED)

F.7.3.1 addAbstractionDefChoice

Description: Adds a choice with the given name and enumerations to the given abstraction definition element.

- Returns: choiceID of type **String** - Handle to a new choice
- Input: abstractionDefinitionID of type **String** - Handle to an abstraction definition element
- Input: name of type **String** - Choice name
- Input: enumerations of type **String[]** - List of enumeration values

F.7.3.2 addAbstractionDefPort

Description: Adds abstractionDefPort with the given logicalName and type to the given abstractionDef element.

- Returns: abstractionDefPortID of type **String** - Handle to a new abstractionDefPort element
- Input: abstractionDefID of type **String** - Handle to an abstractionDef element
- Input: logicalName of type **String** - Logical port name
- Input: type of type **String** - Logical port style (wire or transactional)

F.7.3.3 addAbstractionDefPortMode

Description: Adds abstractionDefPortMode with the given value for the given abstractionDefPort element.

- Returns: abstractionDefPortModeID of type **String** - Handle to a new abstractionDefPortMode element
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element
- Input: mode of type **String** - Logical port mode (onInitiator, onTarget, onSystem)

F.7.3.4 addAbstractionDefPortPacket

Description: Adds a port packet to the given logical port element.

- Returns: portPacketTypeID of type **String** - Handle to the added portPacketType
- Input: absDefPortID of type **String** - Handle to an absDefPort element
- Input: packetName of type **String** - Name of the packet
- Input: packetFieldName of type **String** - Name of the packetField
- Input: packetFieldWidth of type **String** - Width of the packetField

F.7.3.5 addAbstractionDefPortTransactionalOnSystem

Description: Adds an onSystem element to the given port transactional.

- Returns: onSystemID of type **String** - Handle to the added OnSystem
- Input: portID of type **String** - Handle to a port element
- Input: group of type **String** - Group name

F.7.3.6 addAbstractionDefPortWireOnSystem

Description: Adds an onSystem to the given port element.

- Returns: onSystemID of type **String** - Handle to the added onSystem
- Input: portID of type **String** - Handle to a port element
- Input: group of type **String** - Group name

F.7.3.7 addModeConstraintsTimingConstraint

Description: Adds timingConstraint with the given type for the given abstractionDefPortMode element.

- Returns: timingConstraintID of type **String** - Handle to the added timingConstraint
- Input: modeConstraintsID of type **String** - Handle to modeConstraints element
- Input: value of type **float** - The timingConstraint value
- Input: clockName of type **String** - The timingConstraint clock name

F.7.3.8 addPacketPacketField

Description: Adds a packetField for the given packet element.

- Returns: packetFieldID of type **String** - Handle to the added PortPacketField
- Input: packetID of type **String** - Handle to a packet element
- Input: fieldName of type **String** - Name of the field created
- Input: width of type **Long** - Width of the field created

F.7.3.9 removeAbstractionDefChoice

Description: Removes the given choice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: choiceID of type **String** - Handle to a choice element

F.7.3.10 removeAbstractionDefExtends

Description: Removes extends from the given abstractionDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefID of type **String** - Handle to an abstractionDef element

F.7.3.11 removeAbstractionDefPort

Description: Removes the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element

F.7.3.12 removeAbstractionDefPortMatch

Description: Removes match element from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element

F.7.3.13 removeAbstractionDefPortMode

Description: Removes abstractionDefPortMode with the given abstractionDefPortModeID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.14 removeAbstractionDefPortTransactionalModeBusWidth

Description: Removes busWidth from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.15 removeAbstractionDefPortTransactionalModeInitiative

Description: Removes initiative from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPort element

F.7.3.16 removeAbstractionDefPortTransactionalModeKind

Description: Removes kind from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPort element

F.7.3.17 removeAbstractionDefPortTransactionalModePresence

Description: Removes presence from the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.18 removeAbstractionDefPortTransactionalModeProtocol

Description: Removes protocol from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.19 removeAbstractionDefPortTransactionalOnInitiator

Description: Removes the onInitiator on an abstractionDefinition port transactional.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - The identifier of a transactional abstractionDefinition port

F.7.3.20 removeAbstractionDefPortTransactionalOnSystem

Description: Removes the given onSystem element from its containing logical transactional port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: onSystemID of type **String** - Handle to an onSystem element

F.7.3.21 removeAbstractionDefPortTransactionalOnTarget

Description: Removes the onTarget on an abstractionDefinition port transactional.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - the identifier of a transactional abstractionDefinition port

F.7.3.22 removeAbstractionDefPortTransactionalQualifier

Description: Removes the given qualifier from its contained abstractDefinition port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - Handle to qualifier

F.7.3.23 removeAbstractionDefPortWireDefaultValue

Description: Removes defaultValue from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element

F.7.3.24 removeAbstractionDefPortWireModeDirection

Description: Removes direction from the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.25 removeAbstractionDefPortWireModeMirroredModeConstraints

Description: Removes mirroredModeConstraints for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.26 removeAbstractionDefPortWireModeModeConstraints

Description: Removes modeConstraints for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.27 removeAbstractionDefPortWireModePresence

Description: Removes presence from the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.28 removeAbstractionDefPortWireModeWidth

Description: Removes width from the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.29 removeAbstractionDefPortWireOnInitiator

Description: Removes the onInitiator element from a wire abstractionDefinition port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to the abstractionDefPort element

F.7.3.30 removeAbstractionDefPortWireOnSystem

Description: Removes the onSystem element from its containing logical wire port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: onSystemID of type **String** - Handle to an onSystem element

F.7.3.31 removeAbstractionDefPortWireOnTarget

Description: Removes the onTarget element from a wire abstractionDefinitionPort.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to a logical port element

F.7.3.32 removeAbstractionDefPortWireQualifier

Description: Removes the given qualifier from a wire abstractionDefinitionPort.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - Handle to qualifier

F.7.3.33 removeAbstractionDefPortWireRequiresDriver

Description: Removes requiresDriver from a wire abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element

F.7.3.34 removeCustomAttribute

Description: Removes custom attribute of kind from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element

F.7.3.35 removeMandatoryAttribute

Description: Removes mandatory attribute of payloadExtension from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPort element

F.7.3.36 removeModeConstraintsDriveConstraint

Description: Removes driveConstraint with the given modeConstraints.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeConstraintsID of type **String** - Handle to modeConstraints element

F.7.3.37 removeModeConstraintsLoadConstraint

Description: Removes loadConstraint with the given modeConstraints.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeConstraintsID of type **String** - Handle to modeConstraints element

F.7.3.38 removeModeConstraintsTimingConstraint

Description: Removes timingConstraint with the given timingConstraintID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: timingConstraintID of type **String** - Handle to timingConstraint element

F.7.3.39 removePacketEndianness

Description: Removes the endianness associated with the given packet element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetID of type **String** - Handle to a packet element

F.7.3.40 removePacketFieldEndianness

Description: Removes the endianness associated with the given packetField element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.3.41 removePacketFieldQualifier

Description: Removes the given qualifier from its contained packetField.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetFieldID of type **String** - Handle to packetField element

F.7.3.42 removePacketFieldValue

Description: Removes the value of the given packetField.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.3.43 removePacketPacketField

Description: Removes the given packetField from its containing packet element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetFieldID of type **String** - Handle to a packetField element

F.7.3.44 setAbstractionDefExtends

Description: Sets extends with the given value for the given abstractionDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefID of type **String** - Handle to an abstractionDef element
- Input: abstractionDefVLNV of type **String[]** - AbstractionDef VLNV

F.7.3.45 setAbstractionDefPortLogicalName

Description: Sets the logicalName to the given value for the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort elemen
- Input: value of type **String** - new logical port name

F.7.3.46 setAbstractionDefPortMatch

Description: Sets match with the given value for the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element
- Input: match of type **Boolean** - match value

F.7.3.47 setAbstractionDefPortMode

Description: Sets the mode with the given value for the given abstractionDefPort element.

- Returns: modeID of type **String** - Handle of new created mode
- Input: abstractionDefPortID of type **String** - Handle to an abstractionDefPort element
- Input: mode of type **String** - mode value

F.7.3.48 setAbstractionDefPortTransactionalModeBusWidth

Description: Sets busWidth to the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element
- Input: width of type **String** - width expression

F.7.3.49 setAbstractionDefPortTransactionalModeInitiative

Description: Sets initiative for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element
- Input: initiative of type **String** - initiative value. Can be one of 'requires', 'provides' or 'both'

F.7.3.50 setAbstractionDefPortTransactionalModeKind

Description: Sets kind for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element
- Input: kind of type **String** - kind value. Can be one of 'tlm_port', 'tlm_socket', 'simple_socket', 'multi_socket', 'custom'

F.7.3.51 setAbstractionDefPortTransactionalModePresence

Description: Sets presence for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element
- Input: presence of type **String** - presence value. Can be one of 'required', 'illegal' or 'optional'

F.7.3.52 setAbstractionDefPortTransactionalModeProtocol

Description: Sets protocol for the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element

- Input: protocolType of type **String** - the type of protocol. Can be one of 'tlm' or 'custom'

F.7.3.53 setAbstractionDefPortTransactionalOnInitiator

Description: Sets the onInitiator on an abstractionDefinition port transactional.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - the identifier of a transactional abstractionDefinition port

F.7.3.54 setAbstractionDefPortTransactionalOnTarget

Description: Sets the onTarget on an abstractionDefinition port transactional.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - the identifier of a transactional abstractionDefinition port

F.7.3.55 setAbstractionDefPortTransactionalQualifier

Description: Sets the qualifier on an abstractionDefinition port transactional.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portID of type **String** - Handle to port

F.7.3.56 setAbstractionDefPortWire

Description: Set the wire element of an abstraction definition port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstraction definition port

F.7.3.57 setAbstractionDefPortTransactional

Description: Set the transactional element of an abstraction definition port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstraction definition port

F.7.3.58 setAbstractionDefPortWireDefaultValue

Description: Sets defaultValue with the given value for the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element
- Input: valueExpression of type **String** - Logical port default value

F.7.3.59 setAbstractionDefPortWireModeDirection

Description: Sets direction with the given value for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element
- Input: direction of type **String** - Logical port direction

F.7.3.60 setAbstractionDefPortWireModeMirroredModeConstraints

Description: Sets mirroredModeConstraints element for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element
- Input: timingConstraintClockName of type **String** - clockName of the timingConstraint

F.7.3.61 setAbstractionDefPortWireModeModeConstraints

Description: Sets modeConstraints for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortModeID of type **String** - Handle to abstractionDefPortMode element
- Input: timingConstraintClockName of type **String** - clockName of the timingConstraint

F.7.3.62 setAbstractionDefPortWireModePresence

Description: Sets the presence on a wire port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portModeID of type **String** - Handle to a portModeElement (onInitiator, onWire, onTarget)
- Input: presence of type **String** - the presence value

F.7.3.63 setAbstractionDefPortWireModeWidth

Description: Sets the width on a wire port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portModeID of type **String** - Handle to a portModeElement (onInitiator, onWire, onTarget)
- Input: width of type **String** - the presence value

F.7.3.64 setAbstractionDefPortWireOnInitiator

Description: Sets the onInitiator on a wire abstractionDefinition port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to the abstractionDefPort

F.7.3.65 setAbstractionDefPortWireOnTarget

Description: Sets the onTarget element defined on the given logical port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to a logical port element

F.7.3.66 setAbstractionDefPortWireQualifier

Description: Sets the qualifier on the given abstractDefinition port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to port

F.7.3.67 setAbstractionDefPortWireRequiresDriver

Description: Sets requiresDriver with the given value for the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefPortID of type **String** - Handle to abstractionDefPort element
- Input: value of type **Boolean** - Logical port requiresDriver value

F.7.3.68 setModeConstraintsDriveConstraint

Description: Adds driveConstraint with the given type for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeConstraintsID of type **String** - Handle to modeConstraints element
- Input: cellType of type **String** - cellFunction or cellClass of the driveConstraint

F.7.3.69 setModeConstraintsLoadConstraint

Description: Adds loadConstraint with the given type for the given abstractionDefPortMode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeConstraintsID of type **String** - Handle to modeConstraints element
- Input: cellType of type **String** - cellFunction or cellClass of the driveConstraint

F.7.3.70 setOnSystemGroup

Description: Sets the group on an onSystem element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: onSystemID of type **String** - Handle to an onSystem element
- Input: group of type **String** - The group value

F.7.3.71 setPacketEndianness

Description: Sets the endianness associated with the given packet element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetID of type **String** - Handle to a packet element
- Input: endianness of type **String** - Required endianness (“big” or “little”)

F.7.3.72 setPacketFieldEndianness

Description: Sets the endianness associated with the given packetField element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetFieldID of type **String** - Handle to a packetField element
- Input: endianness of type **String** - Required endianness (“big” or “little”)

F.7.3.73 setPacketFieldQualifier

Description: Sets the qualifier on the given packetField.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetFieldID of type **String** - Handle to packetFied

F.7.3.74 setPacketFieldValue

Description: Sets the value of the given packetField.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetFieldID of type **String** - Handle to a packetField element
- Input: expression of type **String** - The new value or expression

F.7.3.75 setPacketFieldWidth

Description: Sets the expression of the “width” field associated with the given packetField element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetFieldID of type **String** - Handle to a packetField element
- Input: expression of type **String** - Handle to the new value

F.7.4 Abstractor (BASE)

F.7.4.1 getAbstractorAbstractorGeneratorIDs

Description: Returns the handles to the generators defined on the given abstractor or abstractorInstance element.

- Returns: generatorIDs of type **String** - List of handles to the generator elements
- Input: abstractorOrAbstractorInstanceID of type **String** - Handle to an abstractor or abstractorInstance element

F.7.4.2 getAbstractorAbstractorInterfaceIDs

Description: Returns the handles to all the abstractorInterfaces defined on the given abstractor or abstractorInstance element.

- Returns: abstractorBusInterfaceIDs of type **String** - List of abstractorBusInterface handles
- Input: abstractorID of type **String** - Handle to an abstractor object

F.7.4.3 getAbstractorAbstractorMode

Description: Returns the abstractorMode defined on the given abstractor object.

- Returns: mode of type **String** - The abstractorMode value
- Input: abstractorID of type **String** - Handle to an abstractor object

F.7.4.4 getAbstractorAbstractorModelID

Description: Returns the handle to the abstractorMode defined on the given abstractor element.

- Returns: abstractorModeID of type **String** - Handle to the abstractorMode element
- Input: abstractorID of type **String** - Handle to an abstractor element

F.7.4.5 getAbstractorBusTypeRefByVLNV

Description: Returns the busType VLNV defined on the given abstractor object.

- Returns: VLNV of type **String** - The VLNV of the referenced busDefinition object
- Input: abstractorID of type **String** - Handle to an abstractor object

F.7.4.6 getAbstractorChoiceIDs

Description: Returns the handles to all the choices defined on the given abstractor or abstractorInstance element.

- Returns: choiceIDs of type **String** - List of handles to choice elements
- Input: abstractorID of type **String** - Handle to an abstractor object

F.7.4.7 getAbstractorComponentInstantiationIDs

Description: Returns the handles to all the componentInstantiations defined on the given abstractor or abstractorInstance element.

- Returns: componentInstantiationIDs of type **String** - List of handles to the componentInstantiation elements
- Input: abstractorOrAbstractorInstanceID of type **String** - Handle to an abstractor or abstractorInstance element

F.7.4.8 getAbstractorFileSetIDs

Description: Returns the handles to all the fileSets defined on the given abstractor or abstractorInstance element.

- Returns: fileSetIDs of type **String** - List of handles to the fileSet elements
- Input: abstractorID of type **String** - Handle to an abstractor object

F.7.4.9 getAbstractorInterfaceAbstractionTypeIDs

Description: Returns the handles to all the abstractionTypes defined on the given abstractorInterface element on an abstractor

- Returns: abstractionTypeIDs of type **String** - List of abstractionType handles
- Input: abstractorInterfaceID of type **String** - Handle to an abstractorInterface element

F.7.4.10 getAbstractorPortIDs

Description: Returns the handles to all the ports defined on the given abstractor or abstractorInstance element.

- Returns: portIDs of type **String** - List of handles to the port elements
- Input: abstractorID of type **String** - Handle to an abstractor object

F.7.4.11 getAbstractorViewIDs

Description: Returns the handles to all the abstractorViews defined on the given abstractor or abstractorInstance element.

- Returns: abstractorViewIDs of type **String** - List of handles to the abstractorView elements
- Input: abstractorID of type **String** - Handle to an abstractor object

F.7.5 Abstractor (EXTENDED)

F.7.5.1 addAbstractorAbstractorGenerator

Description: Adds a generator with the given name and path to the given abstractor element.

- Returns: generatorID of type **String** - Handle to a new generator

- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - Generator name
- Input: generatorExecutable of type **String** - Path to generator executable

F.7.5.2 addAbstractorChoice

Description: Adds a choice with the given name and enumerations to the given abstractor element.

- Returns: choiceID of type **String** - Handle to a new choice
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - Choice name
- Input: enumerations of type **String[]** - List of enumerations

F.7.5.3 addAbstractorComponentInstantiation

Description: Adds a componentInstantiation with the given name for the given abstractor element.

- Returns: componentInstantiationID of type **String** - Handle to a new componentInstantiation
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - ComponentInstantiation name

F.7.5.4 addAbstractorFileSet

Description: Adds a fileSet with the given name to the given abstractor element.

- Returns: fileSetID of type **String** - Handle to a new fileSet
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - FileSet name

F.7.5.5 addAbstractorInterfaceAbstractionType

Description: Adds an abstractionType with the given abstractionRef for the given abstractorBusInterface element.

- Returns: abstractionTypeID of type **String** - Handle to a new abstractionType
- Input: abstractorBusInterfaceID of type **String** - Handle to an abstractorBusInterface element
- Input: abstractionRef of type **String[]** - abstractionDef VLNV

F.7.5.6 addAbstractorStructuredInterfacePort

Description: Adds a structured port with the given name and directionOrInitiative for the given abstractor element.

- Returns: portID of type **String** - Handle to a new port
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - Handle to a port by his name
- Input: subPortName of type **String** - The name of the subPort
- Input: structPortTypeDefTypeName of type **String** - The typeName of the structPortTypeDef

F.7.5.7 addAbstractorStructuredStructPort

Description: Adds a structured struct port with the given name and direction for the given abstractor element.

- Returns: portID of type **String** - Handle to a new port
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - Handle to a port by his name
- Input: subPortName of type **String** - The name of the subPort
- Input: structPortTypeDefTypeName of type **String** - The typeName of the structPortTypeDef
- Input: direction of type **String** - Value of the direction

F.7.5.8 addAbstractorStructuredUnionPort

Description: Adds a structured union port with the given name and direction for the given abstractor element.

- Returns: portID of type **String** - Handle to a new port
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - Handle to a port by his name
- Input: structPortTypeDefTypeName of type **String** - The typeName of the structPortTypeDef
- Input: subPortName of type **String** - The name of the subPort
- Input: direction of type **String** - Value of the direction

F.7.5.9 addAbstractorTransactionalPort

Description: Adds a transactional port with the given name and initiative for the given abstractor element.

- Returns: portID of type **String** - Handle to a new port
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - port name
- Input: initiative of type **String** - Port initiative

F.7.5.10 addAbstractorView

Description: Adds an abstractorView with the given name and envIdentifier for the given abstractor element.

- Returns: abstractorViewID of type **String** - Handle to the added abstractorView
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - Abstractor view name

F.7.5.11 addAbstractorWirePort

Description: Adds a wire port with the given name and direction for the given abstractor element.

- Returns: portID of type **String** - Handle to a new port
- Input: abstractorID of type **String** - Handle to an abstractor element
- Input: name of type **String** - port name
- Input: direction of type **String** - Port direction

F.7.5.12 removeAbstractorAbstractorGenerator

Description: Removes the given generator element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.5.13 removeAbstractorAbstractorInterface

Description: Removes the given abstractorInterface.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractorInterfaceID` of type ***String*** - Handle to an abstractorInterface element

F.7.5.14 removeAbstractorChoice

Description: Removes the given choice element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `choiceID` of type ***String*** - Handle to a choice element

F.7.5.15 removeAbstractorComponentInstantiation

Description: Removes the given componentInstantiation element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.5.16 removeAbstractorFileSet

Description: Removes the given fileSet element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.5.17 removeAbstractorInterfaceAbstractionType

Description: Removes the given abstractionType element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element

F.7.5.18 removeAbstractorPort

Description: Removes the given port element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element

F.7.5.19 removeAbstractorView

Description: Removes the given abstractorView element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractorViewID` of type ***String*** - Handle to an abstractorView element

F.7.5.20 removeFileSetRefGroupFileSetRef

Description: Removes the given FileSetRef from its containing FileSetGroup element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileSetRefID` of type ***String*** - Handle to a fileSetRef

F.7.5.21 setAbstractorAbstractorMode

Description: Sets the abstractorMode for the abstractor.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractorID` of type ***String*** - Handle to an abstractor element
- Input: `abstractorMode` of type ***String*** - Mode name. Can be one of: initiator, target, direct or system

F.7.5.22 setAbstractorBusType

Description: Sets the busType of the abstractor.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractorID` of type ***String*** - Handle to an abstractorType element
- Input: `value` of type ***String[]*** - The VLN of the busType

F.7.6 Access Policy (BASE)

F.7.6.1 getFieldAccessPolicyModeRefByID

Description: Returns the modeID defined on the given field access policy element.

- Returns: `modeID` of type ***String*** - Handle to the referenced mode element
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to an field access policy element
- Input: `modeRef` of type ***String*** - Handle to the referenced modeRef element

F.7.7 Access handle (BASE)

F.7.7.1 getAccessHandleForce

Description: Returns the value of the force attribute defined on the given accessHandle element.

- Returns: `force` of type ***Boolean*** - The value of the force attribute
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element

F.7.7.2 getAccessHandleIDs

Description: Returns the handles to all the accesHandles defined on the given accessHandle element.

- Returns: `accesHandleIDs` of type ***String*** - List of handles to the accesHandle elements
- Input: `elementID` of type ***String*** - Handle to an element

F.7.7.3 getAccessHandleIndicesIDs

Description: Returns the handles to all the indices defined on the given accessHandle element.

- Returns: `indicesID` of type ***String*** - List of handles to the indices elements

- Input: accessHandleID of type **String** - Handle to an accessHandle element

F.7.7.4 getAccessHandlePathSegmentIDs

Description: Returns the handles to all the pathSegments defined on the given accessHandle element.

- Returns: pathSegmentIDs of type **String** - List of handles to the pathSegment elements
- Input: accessHandleID of type **String** - Handle to an accessHandle element

F.7.7.5 getAccessHandleSliceIDs

Description: Returns the handles to all the slides defined on the given accessHandle element.

- Returns: sliceIDs of type **String** - List of handles to the slice elements
- Input: accessHandleID of type **String** - Handle to an accessHandle element

F.7.7.6 getAccessHandleViewRefIDs

Description: Returns the handles to all the views defined on the given accessHandle element.

- Returns: viewIDs of type **String** - List of handles to the view elements
- Input: accessHandleID of type **String** - Handle to an accessHandle element

F.7.8 Access handle (EXTENDED)

F.7.8.1 addAccessHandle

Description: Adds an accessHandle with the given pathSegment to the given element.

- Returns: accessHandleID of type **String** - Handle to a new accessHandle
- Input: elementID of type **String** - Handle to an element
- Input: pathSegment of type **String** - pathSegment value

F.7.8.2 addAccessHandleIndex

Description: Adds an index on an accessHandle of type Port.

- Returns: indexID of type **String** - the index identifier
- Input: accessHandleID of type **String** - Handle to an accessHandle of type Port
- Input: value of type **String** - the index value to set on the index

F.7.8.3 addAccessHandlePathSegment

Description: Adds a pathSegment with the given pathSegmentName to the given AccessHandle element.

- Returns: pathSegmentID of type **String** - Handle to a new pathSegment
- Input: accessHandleID of type **String** - Handle to an accessHandle element
- Input: pathSegment of type **String** - pathSegment name

F.7.8.4 addAccessHandleSlice

Description: Adds a slice with the given pathSegmentValue to the given accessHandle element.

- Returns: sliceID of type **String** - Handle to the added slice
- Input: accessHandleID of type **String** - Handle to accessHandle
- Input: pathSegment of type **String** - Handle to a pathSegment

F.7.8.5 addAccessHandleViewRef

Description: Adds a viewRef with the given name to the given accessHandle element.

- Returns: `viewRefID` of type ***String*** - Handle to a viewRef element
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element
- Input: `viewRef` of type ***String*** - View reference

F.7.8.6 removeAccessHandle

Description: Removes the given accessHandle.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element

F.7.8.7 removeAccessHandleIndex

Description: Removes the given index from its containing port accessHandle indices.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indexID` of type ***String*** - Handle to an index element

F.7.8.8 removeAccessHandlePathSegment

Description: Removes the given pathSegment.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `pathSegmentID` of type ***String*** - Handle to a pathSegment element

F.7.8.9 removeAccessHandleSlice

Description: Removes the given slice.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `sliceID` of type ***String*** - Handle to a slice element

F.7.8.10 removeAccessHandleViewRef

Description: Removes the viewRef with the given name from the given accessHandle element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `viewRefID` of type ***String*** - Handle to a viewxRef element

F.7.8.11 setAccessHandleForce

Description: Sets the given value for the attribute force for the given accessHandle element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element
- Input: `force` of type ***Boolean*** - Value of the attribute force

F.7.9 Access policy (BASE)

F.7.9.1 getAccessPolicyAccess

Description: Returns the access defined on the given accessPolicy element.

- Returns: access of type **String** - The access value
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element

F.7.9.2 getAccessPolicyModeRefByID

Description: Returns the handle to the referenced mode defined on the given accessPolicy element.

- Returns: modeID of type **String** - Handle to the referenced mode element
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element
- Input: modeRef of type **String** - The referenced mode

F.7.9.3 getAccessPolicyModeRefByNames

Description: Returns all the modeRef names defined on the accessPolicy element.

- Returns: modeRef of type **String** - List of the modeRef names
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element

F.7.9.4 getAccessPolicyModeRefIDs

Description: Returns the handles to all the modeRef defined on the given accessPolicy element.

- Returns: modeRefIDs of type **String** - List of handles to the modeRef elements
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element

F.7.9.5 getAccessRestrictionModeRefIDs

Description: Returns the handles to all the modeRefs defined on the given accessRestriction element.

- Returns: modeRefIDs of type **String** - List of handles to the modeRef elements
- Input: accessRestrictionID of type **String** - Handle to an accessRestriction element

F.7.9.6 getAccessRestrictionModeRefbyID

Description: Returns the modeID defined on the given accessRestriction element.

- Returns: modeID of type **String** - Handle to the referenced mode element
- Input: accessRestrictionID of type **String** - Handle to an accessRestriction element
- Input: modeRef of type **String** - Handle to the referenced modeRef element

F.7.9.7 getAccessRestrictionModeRefbyNames

Description: Returns all the modeRef defined on the given accessRestriction element.

- Returns: modeRef of type **String** - The list of all the modeRef values
- Input: accessRestrictionID of type **String** - Handle to an accessRestriction element

F.7.9.8 getAccessRestrictionReadAccessMask

Description: Returns the readAccessMask value defined on the given accessRestriction element.

- Returns: readAccessMask of type **Long** - The readAccessMask value
- Input: accessRestrictionID of type **String** - Handle to an accessRestriction element

F.7.9.9 getAccessRestrictionReadAccessMaskExpression

Description: Returns the readAccessMask expression defined on the given accessRestriction element.

- Returns: `readAccessMask` of type ***String*** - The readAccessMask expression
- Input: `accessRestrictionID` of type ***String*** - Handle to an accessRestriction element

F.7.9.10 getAccessRestrictionReadAccessMaskID

Description: Returns the handle to the readAccessMask defined on the given accessRestriction element.

- Returns: `readAccessMaskID` of type ***String*** - Handle to the readAccessMask element
- Input: `accessRestrictionID` of type ***String*** - Handle to an accessRestriction element

F.7.9.11 getAccessRestrictionWriteAccessMask

Description: Returns the writeAccessMask value defined on the given accessRestriction element.

- Returns: `writeAccessMask` of type ***Long*** - The writeAccessMask value
- Input: `accessRestrictionID` of type ***String*** - Handle to an accessRestriction element

F.7.9.12 getAccessRestrictionWriteAccessMaskExpression

Description: Returns the writeAccessMask expression on an accessRestriction element.

- Returns: `writeAccessMask` of type ***String*** - The writeAccessMask expression
- Input: `accessRestrictionID` of type ***String*** - Handle to an accessRestriction element

F.7.9.13 getAccessRestrictionWriteAccessMaskID

Description: Returns the handle to the readAccessMask element defined on the given accessRestriction element.

- Returns: `writeAccessMaskID` of type ***String*** - Handle to the writeAccessMask element
- Input: `accessRestrictionID` of type ***String*** - Handle to an accessRestriction element

F.7.9.14 getAlternateRegisterAccessPolicyIDs

Description: Returns the handles to all the accessPolicies defined on the given alternateRegister element.

- Returns: `accessPolicyID` of type ***String*** - List of handles to the accessPolicy elements
- Input: `alternateRegisterID` of type ***String*** - Handle to an alternateRegister element

F.7.9.15 getBankAccessPoliciesIDs

Description: Returns the handles to all the accessPolicies defined on the given bank element.

- Returns: `accessPoliciesIDs` of type ***String*** - List of handles to the accessPolicies elements
- Input: `bankID` of type ***String*** - Handle to a bank element

F.7.9.16 getFieldAccessPoliciesFieldAccessPolicyIDs

Description: Returns the handles to all the fieldAccessPolicies defined on the given fieldAccessPolicies element

- Returns: `fieldAccessPolicyIDs` of type ***String*** - List of handles to the fieldAccessPolicy elements
- Input: `fieldAccessPoliciesID` of type ***String*** - Handle to a fieldAccessPolicies element

F.7.9.17 getFieldAccessPolicyAccess

Description: Returns the access defined on the given fieldAccessPolicy element.

- Returns: `access` of type ***String*** - The access value
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a fieldAccessPolicy element

F.7.9.18 getFieldAccessPolicyAccessRestrictionIDs

Description: Returns the handles to all the accessRestrictions defined on the given fieldAccessPolicy element.

- Returns: `accessRestrictionIDs` of type ***String*** - List of handles to the accessRestriction elements
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a fieldAccessPolicy element

F.7.9.19 getFieldAccessPolicyFieldAccessPolicyDefinitionRefByExternalTypeDefID

Description: Returns the handle to the externalTypeDefinitions referenced by the typeDefinitions attribute of the fieldAccessPolicyDefinitionRef defined on the given fieldAccessPolicy element.

- Returns: `fieldAccesspolicyDefinitionRefID` of type ***String*** - Handle to the externalTypeDefinitions element referenced by the typeDefinitions attribute
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a fieldAccessPolicy element

F.7.9.20 getFieldAccessPolicyFieldAccessPolicyDefinitionRefByID

Description: Returns the handle to the fieldAccesspolicyDefinition referenced from the given fieldAccessPolicy element.

- Returns: `fieldAccesspolicyDefinitionRefID` of type ***String*** - Handle to the referenced fieldAccesspolicyDefinition element
- Input: `fieldAccessPolicyID` of type ***String*** - Handle a fieldAccessPolicy element

F.7.9.21 getFieldAccessPolicyFieldAccessPolicyDefinitionRefByName

Description: Returns the fieldAccessPolicyDefinition defined on the given fieldAccessPolicy element.

- Returns: `fieldAccesspolicyDefinition` of type ***String*** - The fieldAccesspolicyDefinition name
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a fieldAccessPolicy element

F.7.9.22 getFieldAccessPolicyFieldAccessPolicyDefinitionRefID

Description: Returns the handle to the fieldAccesspolicyDefinitionRef defined on the given fieldAccessPolicy element.

- Returns: `fieldAccesspolicyDefinitionRefID` of type ***String*** - Handle to the fieldAccesspolicyDefinitionRef element
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a fieldAccessPolicy element

F.7.9.23 getFieldAccessPolicyModeRefByName

Description: Returns all the modeRefs defined on the given fieldAccessPolicy element.

- Returns: `modeRefs` of type ***String*** - List of the referenced modes
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a fieldAccessPolicy element

F.7.9.24 getFieldAccessPolicyModeRefIDs

Description: Returns the handles to all the modeRefs defined on the given fieldAccessPolicy element.

- Returns: modeRefIDs of type **String** - List of handles to the modeRef elements
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy

F.7.9.25 getFieldAccessPolicyModifiedWriteValue

Description: Returns the modifiedWriteValue defined on the given accessPolicy element.

- Returns: modifiedWriteValue of type **String** - The value of the modifiedWriteValue element
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element

F.7.9.26 getFieldAccessPolicyModifiedWriteValueID

Description: Returns the handle to the modifiedWriteValue defined on the given fieldAccessPolicy.

- Returns: modifiedWriteValueID of type **String** - Handle to the modifiedWriteValue element
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.27 getFieldAccessPolicyReadAction

Description: Returns the readAction on a fieldAccessPolicy element.

- Returns: readAction of type **String** - The readAction value
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.28 getFieldAccessPolicyReadActionID

Description: Returns the handle to the readAction defined on the given fieldAccessPolicy element.

- Returns: readActionID of type **String** - Handle to the readAction element
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.29 getFieldAccessPolicyReadResponse

Description: Returns the readResponse value defined on the given fieldAccessPolicy element.

- Returns: readResponse of type **Long** - The readResponse value
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.30 getFieldAccessPolicyReadResponseExpression

Description: Returns the readResponse expression defined on the given fieldAccessPolicy element.

- Returns: readResponse of type **String** - The readResponse expression value
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.31 getFieldAccessPolicyReadResponseID

Description: Returns the handle to the readResponse element defined on the given fieldAccessPolicy element.

- Returns: readResponseID of type **String** - Handle to the readResponse element
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.32 getFieldAccessPolicyReserved

Description: Returns the reserved value on the fieldAccessPolicy element.

- Returns: value of type **Boolean** - The reserved value on the fieldAccessPolicy element
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.33 getFieldAccessPolicyReservedExpression

Description: Returns the reserved expression on the fieldAccessPolicy element.

- Returns: reserved of type **String** - The reserved expression on the fieldAccessPolicy element
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.34 getFieldAccessPolicyReservedID

Description: Returns the handle to the reserved element defined on the given fieldAccessPolicy element.

- Returns: reservedID of type **String** - Handle to the reserved element
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.35 getFieldAccessPolicyTestable

Description: Returns the testable value on the fieldAccessPolicy element.

- Returns: value of type **Boolean** - The testable value
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.36 getFieldAccessPolicyTestableID

Description: Returns the handle to the testable element defined on the given fieldAccessPolicy element.

- Returns: testableID of type **String** - Handle to the testable element
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.37 getFieldAccessPolicyWriteValueConstraintID

Description: Returns the handle to the writeValueConstraint defined on the given fieldAccessPolicy.

- Returns: writeValueConstraintID of type **String** - Handle to the writeValueConstraint element
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.38 getFieldAccesspolicyBroadcastToIDs

Description: Returns the handles to all the broadcastTo elements defined on the given fieldAccessPolicy element.

- Returns: broadcastToIDs of type **String** - Handles to the list of broadcastTo elements
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.9.39 getRegisterAccessPolicyIDs

Description: Returns the handles to all the accessPolicies defined on the given register element.

- Returns: accessPoliciesIDs of type **String** - List of handles to the accessPolicies elements
- Input: registerID of type **String** - Handle to a register element

F.7.9.40 getRegisterFieldFieldAccessPoliciesID

Description: Returns the handle to the fieldAccessPolicies defined on the given registerField element.

- Returns: `fieldAccessPoliciesID` of type ***String*** - Handle to the fieldAccessPolicies element
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.9.41 getTypeDefinitionsFieldAccessPolicyDefinitionIDs

Description: Returns the handles to all the fieldAccessPolicyDefinitions defined on the given typeDefinitions element.

- Returns: `fieldAccessPolicyDefinitionIDs` of type ***String*** - List of handles to the fieldAccessPolicyDefinitionID elements
- Input: `typeDefinitionsID` of type ***String*** - Handle to a typeDefinitions element

F.7.9.42 getWriteValueConstraintMaximum

Description: Returns the maximum value defined on the given writeValueConstraint element.

- Returns: `maximum` of type ***Long*** - The maximum value
- Input: `writeValueConstraintID` of type ***String*** - Handle to a writeValueConstraint element

F.7.9.43 getWriteValueConstraintMaximumExpression

Description: Returns the maximum expression defined on the given writeValueConstraint element.

- Returns: `maximum` of type ***String*** - The maximum expression value
- Input: `writeValueConstraintID` of type ***String*** - Handle to a writeValueConstraint element

F.7.9.44 getWriteValueConstraintMaximumID

Description: Returns the handle to the maximum element defined on the given writeValueConstraint element.

- Returns: `maximumID` of type ***String*** - Handle to the maximum element
- Input: `writeValueConstraintID` of type ***String*** - Handle to a writeValueConstraint element

F.7.9.45 getWriteValueConstraintMinimum

Description: Returns the minimum value defined on the given writeValueConstraint element.

- Returns: `minimum` of type ***Long*** - The minimum value
- Input: `writeValueConstraintID` of type ***String*** - Handle to a writeValueConstraint element

F.7.9.46 getWriteValueConstraintMinimumExpression

Description: Returns the minimum expression defined on the given writeValueConstraint element.

- Returns: `minimum` of type ***String*** - The minimum expression value
- Input: `writeValueConstraintID` of type ***String*** - Handle to a writeValueConstraint element

F.7.9.47 getWriteValueConstraintMinimumID

Description: Returns the handle to the minimum element defined on the given writeValueConstraint element.

- Returns: `minimumID` of type ***String*** - Handle to the minimum element

- Input: `writeValueConstraintID` of type ***String*** - Handle to a `writeValueConstraint` element

F.7.9.48 `getWriteValueConstraintUseEnumeratedValues`

Description: Returns the `UseEnumeratedValues` field on a `writeValueConstraint` element.

- Returns: `value` of type ***Boolean*** - The `UseEnumeratedValues` value
- Input: `writeValueConstraintID` of type ***String*** - Handle to a `writeValueConstraint` element

F.7.9.49 `getWriteValueConstraintWriteAsRead`

Description: Returns the `writeAsRead` value defined on the given `writeValueConstraint` element.

- Returns: `value` of type ***Boolean*** - The `writeAsRead` value
- Input: `writeValueConstraintID` of type ***String*** - Handle to a `writeValueConstraint` element

F.7.10 Access policy (EXTENDED)

F.7.10.1 `addAccessRestrictionModeRef`

Description: Adds a `modeRef` to the given `accessRestriction` element.

- Returns: `modeRefID` of type ***String*** - the `modeRef` identifier
- Input: `accessRestrictionID` of type ***String*** - Handle to an `accessRestriction` element
- Input: `modeRef` of type ***String*** - Name of the referenced mode
- Input: `priority` of type ***Long*** - The mode reference priority

F.7.10.2 `addAddressBlockAccessPolicy`

Description: Adds an `accessPolicy` element on the `addressBlock`.

- Returns: `accessPolicyID` of type ***String*** - Handle to the added `accessPolicy`
- Input: `addressBlockID` of type ***String*** - Handle to an `addressblock`

F.7.10.3 `addAlternateRegisterAccessPolicy`

Description: Adds `accessPolicy` for the given `alternateRegister`.

- Returns: `accessPolicyID` of type ***String*** - Handle to the added `accessPolicy`
- Input: `registerID` of type ***String*** - Handle to an `alternate register`

F.7.10.4 `addExternalTypeDefinitionsResetTypeLink`

Description: Adds a `resetTypeLink` to an `externalTypeDefinitions`.

- Returns: `resetTypeLinkID` of type ***String*** - The identifier of the added `resetTypeLink`
- Input: `externalTypeDefinitionsID` of type ***String*** - Handle to an `externalTypeDefinitions`
- Input: `externalResetTypeReference` of type ***String*** - Handle to the `externalResetTypeReference` to set on the `resetTypeLink`
- Input: `resetTypeReference` of type ***String*** - Handle to the `resetTypeReference` to set on the `resetTypeLink`

F.7.10.5 `addFieldAccessPoliciesFieldAccessPolicy`

Description: Adds an `fieldAccessPolicy` on the given `fieldAccessPolicies` element

- Returns: `fieldAccessPolicyID` of type ***String*** - the `fieldAccessPolicy` identifier
- Input: `fieldAccessPoliciesID` of type ***String*** - Handle to a `fieldAccessPolicies` element

F.7.10.6 addFieldAccessPolicyAccessRestriction

Description: Adds an `accessRestriction` to a `fieldAccessPolicy` element.

- Returns: `accessRestrictionID` of type ***String*** - the `accessRestriction` identifier
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a `fieldAccessPolicy` element

F.7.10.7 addFieldAccessPolicyBroadcastTo

Description: Adds a `broadcast` to the given `fieldAccessPolicy` element.

- Returns: `broadcastToID` of type ***String*** - Handle of the `broadcastTo` element
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a `fieldAccessPolicy` element
- Input: `memoryMapRef` of type ***String*** - Name of the broadcasted `memoryMap`
- Input: `addressBlockRef` of type ***String*** - Name of the broadcasted `addressBlock`
- Input: `registerRef` of type ***String*** - Name of the broadcasted `register`
- Input: `fieldRef` of type ***String*** - Name of the broadcasted `field`

F.7.10.8 addFieldAccessPolicyModeRef

Description: Adds a `modeRef` on the given `fieldAccessPolicy`.

- Returns: `modeRefID` of type ***String*** - Handle to the added `modeRef`
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a `fieldAccessPolicy` element
- Input: `modeRef` of type ***String*** - Name of the referenced mode
- Input: `priority` of type ***Long*** - Priority of the `modeRef`

F.7.10.9 addFieldDefinitionFieldAccessPolicy

Description: Adds an `accessPolicy` to the given `fieldDefinition`.

- Returns: `fieldAccessPolicyID` of type ***String*** - Handle to the added `AccessEntry`
- Input: `fieldDefinitionID` of type ***String*** - Handle to a `fieldDefinition` element

F.7.10.10 addRegisterAccessPolicy

Description: Adds `accessPolicy` on the given `register` element.

- Returns: `accessPolicyID` of type ***String*** - Handle to the added `accessPolicy`
- Input: `registerID` of type ***String*** - Handle to a `register` element

F.7.10.11 addRegister FileAccessPolicy

Description: Adds an `accessPolicy` on a `registerFile` element.

- Returns: `accessPolicyID` of type ***String*** - The `accessPolicy` identifier
- Input: `registerFileID` of type ***String*** - Handle to a `registerFile` element

F.7.10.12 removeAccessPolicyAccess

Description: Removes the `access` field on the given `accessPolicy`.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

- Input: `accessPolicyID` of type ***String*** - Handle to an `accessPolicy` element

F.7.10.13 removeAccessRestrictionModeRef

Description: Removes a modeRef from its containing `accessRestriction` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `modeRefID` of type ***String*** - Name of the referenced mode

F.7.10.14 removeAccessRestrictionReadAccessMask

Description: Removes a `readAccessMask` on an `accessRestriction` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessRestrictionID` of type ***String*** - Handle to an `accessRestriction` element

F.7.10.15 removeAccessRestrictionWriteAccessMask

Description: Removes a `writeAccessMask` on an `accessRestriction` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessRestrictionID` of type ***String*** - Handle to an `accessRestriction` element

F.7.10.16 removeAddressBlockAccessPolicyID

Description: Removes the given `accessPolicy` from its containing `addressBlock` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessPolicyID` of type ***String*** - Handle to an `accessPolicy` element

F.7.10.17 removeAlternateRegisterAccessPolicy

Description: Removes the given `accessPolicy` from its containing `alternateRegister` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessPolicyID` of type ***String*** - Handle to an `accessPolicy`

F.7.10.18 removeFieldAccessPoliciesFieldAccessPolicy

Description: Removes the given `fieldAccessPolicy` from its containing `fieldAccessPolicies` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fieldAccessPoliciesID` of type ***String*** - Handle to an `fieldAccessPolicies` element

F.7.10.19 removeFieldAccessPolicyAccess

Description: Removes the `access` Attribute on the given `fieldAccessPolicy`.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fieldAccessPolicyID` of type ***String*** - Handle to a `fieldAccessPolicy` element

F.7.10.20 removeFieldAccessPolicyAccessRestriction

Description: Removes the given `accessRestriction` from its containing element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessRestrictionID` of type ***String*** - Handle to an `accessRestriction` element

F.7.10.21 removeFieldAccessPolicyBroadcastTo

Description: Removes the given broadcastTo from its containing fieldAccessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to a broadcastTo element

F.7.10.22 removeFieldAccessPolicyFieldAccessPolicyDefinitionRef

Description: Removes the given fieldAccessPolicyDefinitionRef from its containing fieldAccessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.10.23 removeFieldAccessPolicyModeRef

Description: Removes the given modeRef from its containing fieldAccessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeRefID of type **String** - Handle to a modeRef element

F.7.10.24 removeFieldAccessPolicyModifiedWriteValue

Description: Removes the value of the modifiedWriteValue of the given fieldAccessPolicy.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to the fieldAccessPolicy

F.7.10.25 removeFieldAccessPolicyReadAction

Description: Removes the value of the readAction of the given fieldAccessPolicy.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to the fieldAccessPolicy

F.7.10.26 removeFieldAccessPolicyReadResponse

Description: Removes the readResponse field on a fieldAccessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to an fieldAccessPolicy element

F.7.10.27 removeFieldAccessPolicyReserved

Description: Removes the Reserved value for fieldAccessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.10.28 removeFieldAccessPolicyTestable

Description: Removes a testable element on the given fieldAccessPolicy.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.10.29 removeFieldAccessPolicyWriteValueConstraint

Description: Removes the writeValueConstraint elements from the given fieldAccessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

F.7.10.30 removeFieldDefinitionFieldAccessPolicy

Description: Removes the given field accessPolicy from its containing FileDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element

F.7.10.31 removeRegisterAccessPolicy

Description: Removes the given accessPolicy from its containing register element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element

F.7.10.32 removeRegisterFileAccessPolicy

Description: Removes the given accessPolicy from its containing registerFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element

F.7.10.33 setAccessPolicyAccess

Description: Sets the access field on the given accessPolicy.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element
- Input: access of type **String** - Access enumerated value. Can be one of: read-only, write-only, read-write, writeOnce, read-writeOnce or no-access

F.7.10.34 setAccessRestrictionReadAccessMask

Description: Sets a readAccessMask on an accessRestriction element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessRestrictionID of type **String** - Handle to an accessRestriction element
- Input: readAccessMask of type **String** - Value of the readAccessMask

F.7.10.35 setAccessRestrictionWriteAccessMask

Description: Sets a writeAccessMask on an accessRestriction element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessRestrictionID of type **String** - Handle to an accessRestriction element
- Input: writeAccessMask of type **String** - Value of the writeAccessMask

F.7.10.36 setFieldAccessPolicyAccess

Description: Sets the access Attribute on the given fieldAccessPolicy.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element
- Input: access of type **String** - Access enumerated value. Can be one of: read-only, write-only, read-write, writeOnce or no-access

F.7.10.37 setFieldAccessPolicyFieldAccessPolicyDefinitionRef

Description: Sets the FieldAccessPolicyDefinitionRef on the given fieldAccesspolicy.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element
- Input: value of type **String** - Name of the referenced fieldAccessPolicyDefinition in an external typeDefinitions
- Input: typeDefinitions of type **String** - Name of the component externalTypeDefinitions

F.7.10.38 setFieldAccessPolicyModifiedWriteValue

Description: Sets the value of the modifiedWriteValue of the given fieldAccessPolicy.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to the fieldAccessPolicy
- Input: value of type **String** - Value of the modifiedWriteValue. Can be one of: oneToClear, oneToSet, oneToToggle, zeroToClear, zeroToSet, zeroToToggle, clear, set or modify.

F.7.10.39 setFieldAccessPolicyReadAction

Description: Sets the read action field on a fieldAccessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to an fieldAccessPolicy element
- Input: readAction of type **String** - the readAction value to set

F.7.10.40 setFieldAccessPolicyReadResponse

Description: Sets the readResponse field on a fieldAccessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to an fieldAccessPolicy element
- Input: readResponse of type **String** - The readResponse value to set

F.7.10.41 setFieldAccessPolicyReserved

Description: Sets the Reserved value for fieldAccesspolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element
- Input: reserved of type **String** - Reserved value

F.7.10.42 setFieldAccessPolicyTestable

Description: Sets a testable element on the given fieldAccesspolicy.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element

- Input: testable of type **Boolean** - Handle to value of the added testable

F.7.10.43 setFieldAccessPolicyWriteValueConstraintMinMax

Description: Sets a writeValueConstraint with a minimum and maximum on fieldAccessPolicy with all his mandatory attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element
- Input: minimum of type **String** - The minimum value
- Input: maximum of type **String** - The maximum value

F.7.10.44 setFieldAccessPolicyWriteValueConstraintUseEnumeratedValue

Description: Sets a writeValueConstraint with a useEnumeratedValue on fieldAccessPolicy with all his mandatory attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element
- Input: useEnumeratedValue of type **Boolean** - The useEnumeratedValue value

F.7.10.45 setFieldAccessPolicyWriteValueConstraintWriteAsRead

Description: Sets a writeValueConstraint with a writeAsRead on fieldAccessPolicy with all his mandatory attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldAccessPolicyID of type **String** - Handle to a fieldAccessPolicy element
- Input: writeAsRead of type **Boolean** - The writeAsRead value

F.7.10.46 setWriteValueConstraintMaximum

Description: Sets the maximum field on a writeValueConstraint element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: writeValueConstraintID of type **String** - Handle to a writeValueConstraint element
- Input: maximum of type **String** - Maximum value

F.7.10.47 setWriteValueConstraintMinimum

Description: Sets the minimum on a writeValueConstraint element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: writeValueConstraintID of type **String** - Handle to a writeValueConstraint element
- Input: minimum of type **String** - Minimum value

F.7.10.48 setWriteValueConstraintUseEnumeratedValues

Description: Sets the UseEnumeratedValues on a writeValueConstraint element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: writeValueConstraintID of type **String** - Handle to a writeValueConstraint element
- Input: useEnumeratedValues of type **Boolean** - True to only allow write enumeration values to be written

F.7.10.49 setWriteValueConstraintWriteAsRead

Description: Sets the writeAsRead element of the given writeValueConstraint element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: writeValueConstraintID of type **String** - Handle to a writeValueConstraint element
- Input: writeAsRead of type **Boolean** - True if the access is writeAsRead

F.7.11 Address space (BASE)

F.7.11.1 getAddressSpaceAddressUnitBits

Description: Returns the addressUnitBits defined on the given addressSpace element.

- Returns: addressUnitBits of type **Long** - The addressUnitBits value
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.2 getAddressSpaceAddressUnitBitsExpression

Description: Returns the addressUnitBits expression on the given address space element.

- Returns: addressUnitBitsExpression of type **String** - The addressUnitBits expression
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.3 getAddressSpaceAddressUnitBitsID

Description: Returns the handle to the addressUnitBits defined on the given addressSpace element.

- Returns: addressUnitBitsID of type **String** - Handle to the addressUnitBits
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.4 getAddressSpaceLocalMemoryMapID

Description: Returns the handle to the localMemoryMap defined on the given addressSpace element.

- Returns: localMemoryMapID of type **String** - Handle to a localMemoryMap element
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.5 getAddressSpaceRange

Description: Returns the range defined on the given addressSpace element.

- Returns: range of type **Long** - The range value
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.6 getAddressSpaceRangeExpression

Description: Returns the range expression defined on the given addressSpace element.

- Returns: rangeExpression of type **String** - The addressSpace range
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.7 getAddressSpaceRangeID

Description: Returns the handle to the range defined on the given addressSpace element.

- Returns: rangeID of type **String** - Handle to the range

- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.8 getAddressSpaceSegmentIDs

Description: Returns the handles to all the segments defined on the given addressSpace element.

- Returns: segmentIDs of type **String** - List of handles to the segment elements
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.9 getAddressSpaceWidth

Description: Returns the width defined on the given addressSpace element.

- Returns: width of type **Long** - The width value
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.10 getAddressSpaceWidthExpression

Description: Returns the width expression defined on the given addressSpace element.

- Returns: widthExpression of type **String** - The addressSpace width
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.11 getAddressSpaceWidthID

Description: Returns the handle to the width defined on the given addressSpace element.

- Returns: widthID of type **String** - Handle to the width
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.12 getAliasOfAddressSpaceRefByName

Description: Returns the addressSpaceRef defined on the given aliasOf element.

- Returns: addressSpaceRef of type **String** - The referenced addressSpace
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.11.13 getAliasOfAddressSpaceRefID

Description: Returns the handle to the addressSpaceRef defined on the given aliasOf element.

- Returns: addressSpaceRefID of type **String** - Handle to the addressSpaceRef element
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.11.14 getLocalMemoryMapAddressBlockIDs

Description: Returns the handles to all the localAddressBlocks defined on the given addressSpace element.

- Returns: localAddressBlockIDs of type **String** - Handle to a localAddressBlock element
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.15 getLocalMemoryMapBankIDs

Description: Returns the handles to all the localBanks defined on the given addressSpace element.

- Returns: localBankIDs of type **String** - List of handles to localBank elements
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.11.16 getRegionAddressOffset

Description: Returns the addressOffset on a region element.

- Returns: `addressOffset` of type ***Long*** - The addressOffset value
- Input: `regionID` of type ***String*** - Handle to a region element

F.7.11.17 getRegionAddressOffsetExpression

Description: Returns the addressOffset expression on a region element.

- Returns: `addressOffset` of type ***String*** - The addressOffset expression
- Input: `regionID` of type ***String*** - Handle to a region element

F.7.11.18 getRegionAddressOffsetID

Description: Returns the handle to the addressOffset defined on the given region element.

- Returns: `addressOffsetID` of type ***String*** - Handle to the addressOffset element
- Input: `regionID` of type ***String*** - Handle to a region element

F.7.11.19 getRegionRange

Description: Returns the range defined on the given region element.

- Returns: `rangeValue` of type ***Long*** - The range value
- Input: `regionID` of type ***String*** - Handle to a region element

F.7.11.20 getRegionRangeExpression

Description: Returns the range expression defined on the given region element.

- Returns: `range` of type ***String*** - The range expression
- Input: `regionID` of type ***String*** - Handle to a region element

F.7.11.21 getRegionRangeID

Description: Returns the handle to the range defined on the given region element.

- Returns: `rangeID` of type ***String*** - Handle to the range element
- Input: `regionID` of type ***String*** - Handle to a region element

F.7.11.22 getSegmentAddressOffset

Description: Returns the addressOffset defined on the given segment element.

- Returns: `addressOffset` of type ***Long*** - The address offset
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.11.23 getSegmentAddressOffsetExpression

Description: Returns the addressOffset expression defined on the given segment element.

- Returns: `addressOffset` of type ***String*** - The addressOffset expression
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.11.24 getSegmentAddressOffsetID

Description: Returns the handle to the addressOffset defined on the given segment element.

- Returns: `addressOffsetID` of type ***String*** - Handle to the addressOffset
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.11.25 getSegmentRange

Description: Returns the range defined on the given segment element.

- Returns: `range` of type ***Long*** - The range value
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.11.26 getSegmentRangeExpression

Description: Returns the range expression defined on the given segment element.

- Returns: `range` of type ***String*** - The range expression
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.11.27 getSegmentRangeID

Description: Returns the handle to the range defined on the given segment element.

- Returns: `rangeID` of type ***String*** - Handle to the range
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.12 Address space (EXTENDED)

F.7.12.1 addAddressSpaceSegment

Description: Adds a segment with the given name, addressOffset, and range to the given addressSpace element.

- Returns: `segmentID` of type ***String*** - Handle to a new segment element
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element
- Input: `name` of type ***String*** - Segment name
- Input: `addressOffset` of type ***String*** - Segment addressOffset
- Input: `range` of type ***String*** - Segment range

F.7.12.2 addExecutableImageFileSetRef

Description: Adds a fileSetRef with the given localName to the given executableImage element.

- Returns: `fileSetRefID` of type ***String*** - Handle to a new fileSetRef
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element
- Input: `localName` of type ***String*** - fileSetRef localName

F.7.12.3 addLocalMemoryMapAddressBlock

Description: Adds an addressBlock with the given name, baseAddress, range, and width to the given localMemoryMap element.

- Returns: `addressBlockID` of type ***String*** - Handle to a new addressBlock element
- Input: `localMemoryMapID` of type ***String*** - Handle to a localMemoryMap element

- Input: name of type **String** - AddressBlock name
- Input: baseAddress of type **String** - AddressBlock baseAddress
- Input: range of type **String** - AddressBlock range
- Input: width of type **String** - AddressBlock width

F.7.12.4 removeAddressSpaceAddressUnitBits

Description: Removes the addressUnitBits on the given address space element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.12.5 removeAddressSpaceLocalMemoryMap

Description: Removes the localMemoryMap on the given addressSpace element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressSpaceID of type **String** - Handle to an addressSpace element

F.7.12.6 removeAddressSpaceSegment

Description: Removes the given segment element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: segmentID of type **String** - Handle to a segment element

F.7.12.7 removeExecutableImageFileSetRef

Description: Removes the fileSetRef with the given fileSetRefID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileSetRefID of type **String** - Handle to an fileSetRef element

F.7.12.8 removeLinkerCommandFileGenerator

Description: Removes the given generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator element

F.7.12.9 removeLocalMemoryMapAddressBlock

Description: Removes the given addressBlock element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.12.10 removeLocalMemoryMapBank

Description: Removes the given localBank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankID of type **String** - Handle to a localBank element

F.7.12.11 setAddressSpaceAddressUnitBits

Description: Sets the addressUnitBits with the given value to the given address space element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element
- Input: `addressUnitBitsExpression` of type ***String*** - AddressSpace addressUnitBits

F.7.12.12 setAddressSpaceLocalMemoryMap

Description: Adds a localMemoryMap with the given name to the given addressSpace element with a default addressBlock and addressBlockDefinitionRef.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element
- Input: `name` of type ***String*** - addressSpace name
- Input: `addressBlockName` of type ***String*** - addressBlock name
- Input: `baseAddress` of type ***String*** - baseAddress expression for the addressBlock
- Input: `range` of type ***String*** - addressBlock range
- Input: `width` of type ***String*** - addressBlock width

F.7.12.13 setAddressSpaceRange

Description: Sets the range expression of the addressSpace.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element
- Input: `value` of type ***String*** - The value of the range expression

F.7.12.14 setAddressSpaceWidth

Description: Sets the width expression of the addressSpace.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element
- Input: `value` of type ***String*** - Width value expression

F.7.12.15 setSegmentAddressOffset

Description: Sets the address offset of the segment.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `segmentID` of type ***String*** - Handle to a segment element
- Input: `value` of type ***String*** - Value of the address offset

F.7.12.16 setSegmentRange

Description: Sets the range expression of the segment.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `segmentID` of type ***String*** - Handle to a segment element
- Input: `value` of type ***String*** - Value of the range

F.7.13 Array (BASE)

F.7.13.1 getArrayDimIDs

Description: Returns the handles to all the dim elements defined on the given register or registerField array element.

- Returns: `dimID` of type ***String*** - List of handles to the dim elements
- Input: `arrayOrfieldArrayID` of type ***String*** - Handle to an array element

F.7.13.2 getArrayIDs

Description: Returns arrayIDs of the given element.

- Returns: `arrayIDs` of type ***String*** - List of array handles
- Input: `arrayContainerElementID` of type ***String*** - Handle to an element that has an array element

F.7.13.3 getArrayLeftID

Description: Returns the handle to the left range defined on the given array element.

- Returns: `leftID` of type ***String*** - Handle to the left range
- Input: `arrayID` of type ***String*** - Handle to an array element

F.7.13.4 getArrayRange

Description: Returns the range defined on the given array element.

- Returns: `range` of type ***Long*** - Array of two range values: left and right
- Input: `arrayID` of type ***String*** - Handle to an array element

F.7.13.5 getArrayRangeExpression

Description: Returns the range expressions defined on the given array element.

- Returns: `rangeExpression` of type ***String*** - Array of two range expressions: left and right
- Input: `arrayID` of type ***String*** - Handle to an array element

F.7.13.6 getArrayRightID

Description: Returns the handle to the right range defined on the given array element.

- Returns: `rightID` of type ***String*** - Handle to the right range
- Input: `arrayID` of type ***String*** - Handle to an array element

F.7.13.7 getArrayStride

Description: Returns the stride value defined on the given array element.

- Returns: `strideValue` of type ***Long*** - The stride value
- Input: `arrayID` of type ***String*** - Handle to an array element

F.7.13.8 getArrayStrideExpression

Description: Returns the stride expression defined on the given array element.

- Returns: `strideExpression` of type ***String*** - The stride expression

- Input: `arrayID` of type ***String*** - Handle to an array element

F.7.13.9 `getArrayStrideID`

Description: Returns the handle to the stride defined on the given register or registerField array element.

- Returns: `stride` of type ***String*** - List of handles to the stride or bitStride elements
- Input: `arrayOrfieldArrayID` of type ***String*** - Handle to an array element

F.7.13.10 `getDimExpression`

Description: Returns the dim expression defined on the given dim element.

- Returns: `dim` of type ***String*** - The dim expression value
- Input: `dimID` of type ***String*** - Handle to a dim element

F.7.13.11 `getDimIndexVar`

Description: Returns the name of the indexVar attribute defined on the given Dim element.

- Returns: `indexVar` of type ***String*** - The index variable
- Input: `dimID` of type ***String*** - Handle to a Dim element

F.7.13.12 `getDimValue`

Description: Returns the dim value defined on the given dim element.

- Returns: `dim` of type ***Long*** - The dim value
- Input: `dimID` of type ***String*** - Handle to a dim element

F.7.13.13 `getRegisterArrayID`

Description: Returns the handle to the array defined on the given register.

- Returns: `arrayID` of type ***String*** - Handle to the register array element
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.13.14 `getRegisterFieldArrayID`

Description: Returns the handle to the array defined on the given register field.

- Returns: `arrayID` of type ***String*** - Handle to the field array element
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.14 Array (EXTENDED)

F.7.14.1 `addArray`

Description: Adds an array to the given element.

- Returns: `arrayID` of type ***String*** - Handle to new array
- Input: `arrayContainerElementID` of type ***String*** - Handle to an element that has an array element
- Input: `range` of type ***String[]*** - Range expression with left in index 0 and right in index 1

F.7.14.2 addArrayDim

Description: Adds a dim element to an array on a register or a registerField

- Returns: dimID of type **String** - Handle to the added dim
- Input: arrayOrfieldArrayID of type **String** - Handle to an array on a register or a registerField
- Input: value of type **String** - the value of the dim element

F.7.14.3 removeArray

Description: Removes the given array.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: arrayID of type **String** - Handle to an array element

F.7.14.4 removeArrayDim

Description: Removes the given dimension from its containing array element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: dimID of type **String** - Handle to a dim element

F.7.14.5 removeArrayStride

Description: Removes the stride from the given array element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: arrayID of type **String** - Handle to an array element

F.7.14.6 removeIndexVarAttribute

Description: Removes the indexVar attribute of the given dim element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: dimID of type **String** - Handle to a dim element

F.7.14.7 setArrayStride

Description: Sets the stride value from to the given array element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: arrayID of type **String** - Handle to an array element
- Input: stride of type **String** - The stride value to set

F.7.14.8 setDimIndexVar

Description: Sets the indexVar attribute of the given Dim element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: dimID of type **String** - Handle to a Dim element
- Input: value of type **String** - New value expression

F.7.15 Assertion (BASE)

F.7.15.1 getAssertionAssert

Description: Returns the value of the given assertion element.

- Returns: value of type **Boolean** - The assertion value
- Input: assertionID of type **String** - Handle to an assertion element

F.7.15.2 getAssertionAssertExpression

Description: Returns the expression of the given assertion element.

- Returns: expression of type **String** - The assertion expression
- Input: assertionID of type **String** - Handle to an assertion element

F.7.15.3 getAssertionAssertID

Description: Returns the handle to the assert defined on the given assertion element.

- Returns: assertID of type **String** - Handle to the assert element
- Input: assertionID of type **String** - Handle to an assertion element

F.7.15.4 getAssertionIDs

Description: Returns the handles to all the assertions defined on the given element.

- Returns: assertionIDs of type **String** - List of handles to assertion elements
- Input: assertionContainerElementID of type **String** - Handle to an element that has assertion elements

F.7.16 Assertion (EXTENDED)

F.7.16.1 addAssertion

Description: Adds an assertion with the given name and given value to the given element.

- Returns: assertionID of type **String** - Handle to a new assertion
- Input: assertionContainerElementID of type **String** - Handle to an element that has assertion elements
- Input: name of type **String** - Assertion name
- Input: expression of type **String** - Assertion expression

F.7.16.2 removeAssertion

Description: Removes the given assertion.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: assertionID of type **String** - Handle to an assertion element

F.7.16.3 setAssertionAssert

Description: Sets the assert of the given assertion.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: assertionID of type **String** - Handle to an assertion element

- Input: assertExpression of type **String** - Assert expression

F.7.17 Bus definition (BASE)

F.7.17.1 getBusDefinitionBroadcast

Description: Returns the broadcast element defined on the given busDef or busDefInstance element.

- Returns: value of type **Boolean** - The bus broadcast value
- Input: busDefOrBusDefInstanceID of type **String** - Handle to a busDef or busDefInstance element

F.7.17.2 getBusDefinitionChoiceIDs

Description: Returns the handles to all the choices defined on the given bus definition object element.

- Returns: choiceIDs of type **String** - List of handles to choice elements
- Input: busDefinitionID of type **String** - Handle to an bus definition object element

F.7.17.3 getBusDefinitionDirectConnection

Description: Returns the directConnection element defined on the given busDef or busDefInstance element.

- Returns: value of type **Boolean** - The bus directionConnection value
- Input: busDefOrBusDefInstanceID of type **String** - Handle to a busDef or busDefInstance element

F.7.17.4 getBusDefinitionExtendsRefByVLAN

Description: Returns the extended VLVN defined on the given busDefinition object.

- Returns: VLVN of type **String** - The VLVN of the extended busDefinition object
- Input: busDefinitionID of type **String** - Handle to a busDefinition object

F.7.17.5 getBusDefinitionIsAddressable

Description: Returns the isAddressable element defined on the given busDef or busDefInstance element.

- Returns: value of type **Boolean** - The bus isAddressable value
- Input: busDefOrBusDefInstanceID of type **String** - Handle to a busDef or busDefInstance element

F.7.17.6 getBusDefinitionMaxInitiators

Description: Returns the maxInitiators defined on the given busDef or busDefInstance element.

- Returns: value of type **Long** - The maximum number of initiators
- Input: busDefOrBusDefInstanceID of type **String** - Handle to a busDef or busDefInstance element

F.7.17.7 getBusDefinitionMaxInitiatorsExpression

Description: Returns the maxInitiators expression defined on the given busDef or busDefInstance element.

- Returns: value of type **String** - The maxInitiators expression
- Input: busDefOrBusDefInstanceID of type **String** - Handle to a busDef or busDefInstance element

F.7.17.8 getBusDefinitionMaxInitiatorsID

Description: Returns the maximum number of initiators defined on the given busDefinition element.

- Returns: value of type ***String*** - The maximum number of initiators that can connect on this bus
- Input: busDefID of type ***String*** - Handle to a busDef element

F.7.17.9 getBusDefinitionMaxTargets

Description: Returns the maxTargets defined on the given busDef or busDefInstance element.

- Returns: value of type ***Long*** - The maximum number of targets
- Input: busDefOrBusDefInstanceID of type ***String*** - Handle to a busDef or busDefInstance element

F.7.17.10 getBusDefinitionMaxTargetsExpression

Description: Returns the maxTargets expression defined on the given busDefinition element.

- Returns: value of type ***String*** - The maxTargets expression
- Input: busDefinitionID of type ***String*** - Handle to a busDefinition element

F.7.17.11 getBusDefinitionMaxTargetsID

Description: Returns the maximum number of targets defined on the given busDefinition element.

- Returns: value of type ***String*** - The maximum number of targets that can connect on this bus
- Input: busDefID of type ***String*** - Handle to a busDef element

F.7.17.12 getBusDefinitionSystemGroupNamesIDs

Description: Returns the handles to all the systemGroupNames defined on the given busDefinition element.

- Returns: systemGroupNameIDs of type ***String*** - List of handles to the systemGroupName elements
- Input: busDefinitionID of type ***String*** - Handle to a busDefinition element

F.7.18 Bus definition (EXTENDED)

F.7.18.1 addBusDefinitionChoice

Description: Adds a choice with the given name and enumerations to the given bus definition element.

- Returns: choiceID of type ***String*** - Handle to a new choice
- Input: busDefinitionID of type ***String*** - Handle to a bus definition element
- Input: name of type ***String*** - Choice name
- Input: enumerations of type ***String[]*** - List of enumeration values

F.7.18.2 addBusDefinitionSystemGroupName

Description: Adds the given system group name to the list of busDefinition systemGroupNames element.

- Returns: systemGroupNameID of type ***String*** - The systemGroupName identifier
- Input: busDefinitionID of type ***String*** - Handle of a busDefinition element
- Input: value of type ***String*** - The systemGroupName value

F.7.18.3 removeBusDefinitionBroadcast

Description: Removes broadcast from the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element

F.7.18.4 removeBusDefinitionChoice

Description: Removes the given choice element from the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: choiceID of type **String** - Handle to a choice element

F.7.18.5 removeBusDefinitionExtends

Description: Removes extends from the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element

F.7.18.6 removeBusDefinitionMaxInitiators

Description: Removes maxInitiators for the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element

F.7.18.7 removeBusDefinitionMaxTargets

Description: Removes maxTargets for the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element

F.7.18.8 setBusDefinitionBroadcast

Description: Sets broadcast with the given value for the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element
- Input: value of type **Boolean** - BusDef broadcast

F.7.18.9 setBusDefinitionDirectConnection

Description: Sets directionConnection with the given value for the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element
- Input: value of type **Boolean** - BusDef directionConnection

F.7.18.10 setBusDefinitionExtends

Description: Sets extends with the given value for the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element

- Input: busDefVLNV of type **String[]** - BusDef VLNV

F.7.18.11 setBusDefinitionIsAddressable

Description: Sets isAddressable with the given value for the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element
- Input: value of type **Boolean** - BusDef isAddressable

F.7.18.12 setBusDefinitionMaxInitiators

Description: Sets maxInitiators with the given value for the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element
- Input: valueExpression of type **String** - BusDef maxInitiators

F.7.18.13 setBusDefinitionMaxTargets

Description: Sets maxTargets with the given value for the given busDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busDefID of type **String** - Handle to a busDef element
- Input: valueExpression of type **String** - BusDef maxTargets

F.7.19 Bus interface (BASE)

F.7.19.1 getAbstractionTypeAbstractionRefByID

Description: Returns the handle to the abstractionDefinition instance referenced from the given abstractionType element

- Returns: abstractionDefinitionID of type **String** - Handle to the referenced abstraction-Definition object
- Input: abstractionTypeID of type **String** - Handle to an abstractionType element

F.7.19.2 getAbstractionTypeAbstractionRefByVLNV

Description: Returns the VLNV of the abstractionDefinition referenced from the given abstractionType element.

- Returns: VLNV of type **String** - The VLNV of the referenced abstractionDefinition object
- Input: abstractionTypeID of type **String** - Handle to an abstractionType element

F.7.19.3 getAbstractionTypePortMapIDs

Description: Returns the handles to all the portMaps defined on the given abstractionType element.

- Returns: portMapIDs of type **String** - List of handles to the portMap elements
- Input: abstractionTypeID of type **String** - Handle to an abstractionType element

F.7.19.4 getAbstractionTypeViewRefByID

Description: Returns all the viewRefs defined on the given abstractionType element.

- Returns: `viewID` of type ***String*** - Handle to the referenced view
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element
- Input: `viewRef` of type ***String*** - Handle to the viewRef element

F.7.19.5 getAbstractionTypeViewRefByNames

Description: Returns all the viewRefs defined on the given abstractionType element.

- Returns: `viewRefs` of type ***String*** - List of the referenced views
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element

F.7.19.6 getAbstractionTypeViewRefIDs

Description: Returns the handles to all the viewRefs defined on the given abstractionType element.

- Returns: `viewRefsIDs` of type ***String*** - List of handles to the viewRef elements
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element

F.7.19.7 getAddressSpaceRefBaseAddress

Description: Returns the baseAddress resolved value on an addressSpaceRef element

- Returns: `baseAddressValue` of type ***Long*** - The resolved baseAddress value
- Input: `addressSpaceRefID` of type ***String*** - Handle to an addressSpaceRef element

F.7.19.8 getAddressSpaceRefBaseAddressExpression

Description: Returns the baseAddress expression on an addressSpaceRef element

- Returns: `baseAddressExpression` of type ***String*** - The baseAddress expression
- Input: `addressSpaceRefID` of type ***String*** - Handle to an addressSpaceRef element

F.7.19.9 getAddressSpaceRefBaseAddressID

Description: Returns the handle to the baseAddress defined on the given addressSpaceRef element.

- Returns: `baseAddressID` of type ***String*** - Handle to the baseAddress element
- Input: `addressSpaceRefID` of type ***String*** - Handle to an addressSpaceRef element

F.7.19.10 getAddressSpaceRefModeRefByID

Description: Returns the handle to the mode referenced from the given addressSpaceRef element.

- Returns: `modeID` of type ***String*** - Handle to the referenced mode element
- Input: `addressSpaceRefID` of type ***String*** - Handle to an addressSpaceRef element
- Input: `modeRef` of type ***String*** - The referenced mode

F.7.19.11 getAddressSpaceRefModeRefByNames

Description: Returns all the modeRefs defined on the given addressSpaceRef element

- Returns: `modeRefs` of type ***String*** - List of the referenced modes
- Input: `addressSpaceRefID` of type ***String*** - Handle to an addressSpaceRef element

F.7.19.12 getBaseAddressesRange

Description: Returns the range value defined on the given baseAddresses element.

- Returns: `rangeValue` of type **Long** - The range value
- Input: `baseAddressesID` of type **String** - Handle to a baseAddresses element

F.7.19.13 getBaseAddressesRangeExpression

Description: Returns the range expression defined on the given baseAddresses element

- Returns: `rangeExpression` of type **String** - The range expression
- Input: `baseAddressesID` of type **String** - Handle to a baseAddresses element

F.7.19.14 getBaseAddressesRangeID

Description: Returns the handle to the range defined on the given baseAddresses element

- Returns: `rangeID` of type **String** - Handle to the range element
- Input: `baseAddressesID` of type **String** - Handle to a baseAddresses element

F.7.19.15 getBaseAddressesRemapAddressIDs

Description: Returns the handles to all the remapAddress defined on the given baseAddresses element

- Returns: `remapAddressIDs` of type **String** - List of handles to the baseAddress elements
- Input: `baseAddressesID` of type **String** - Handle to a baseAddresses element

F.7.19.16 getBusInterfaceAbstractionTypeIDs

Description: Returns the handles to all the abstractionTypes defined on the given busInterface element.

- Returns: `abstractionTypeIDs` of type **String** - List of handles to the abstractionType elements
- Input: `busInterfaceID` of type **String** - Handle to a busInterface element

F.7.19.17 getBusInterfaceBitSteering

Description: Returns the bitSteering defined on the given busInterface element.

- Returns: `value` of type **Boolean** - The bitSteering value
- Input: `busInterfaceID` of type **String** - Handle to a busInterface element

F.7.19.18 getBusInterfaceBitSteeringExpression

Description: Returns the bitSteering expression defined on the given busInterface element.

- Returns: `valueExpression` of type **String** - The bitSteering expression
- Input: `busInterfaceID` of type **String** - Handle to a busInterface element

F.7.19.19 getBusInterfaceBitSteeringID

Description: Returns the handle to the bitSteering defined on the given busInterface element.

- Returns: `bitSteeringID` of type **String** - Handle to the bitSteering element
- Input: `busInterfaceID` of type **String** - Handle to a busInterface element

F.7.19.20 getBusInterfaceBitsInLau

Description: Returns the bitsInLau resolved value on a busInterface element.

- Returns: value of type **Long** - The bitsInLau resolved value
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.19.21 getBusInterfaceBitsInLauExpression

Description: Returns the bitsInLau expression defined on the given busInterface element.

- Returns: bitsInLauExpression of type **String** - The bitsInLau expression
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.19.22 getBusInterfaceBitsInLauID

Description: Returns the handle to the bitsInLau defined on the given busInterface element.

- Returns: bitsInLauID of type **String** - Handle to the bitsInLau element
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.19.23 getBusInterfaceBusTypeID

Description: Returns the handle to the busType defined on the given busInterface element.

- Returns: busTypeID of type **String** - Handle to the busType element
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.19.24 getBusInterfaceBusTypeRefByID

Description: Returns the handle to the busDefinition referenced from the busType defined on the given busInterface element.

- Returns: busDefinitionID of type **String** - Handle to the referenced busDefinition object
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.19.25 getBusInterfaceBusTypeRefByVNV

Description: Returns the VNV of the busType defined on the given busInterface element.

- Returns: busTypeVNV of type **String** - The VNV of the bus
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.19.26 getBusInterfaceConnectionRequired

Description: Returns the connectionRequired defined on the given busInterface element.

- Returns: value of type **Boolean** - The connectionRequired value
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.19.27 getBusInterfaceEndianness

Description: Returns the endianness defined on the given busInterface element.

- Returns: value of type **String** - The endianness value
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.19.28 getBusInterfaceInitiatorID

Description: Returns the handle to the initiator defined on the given busInterface element.

- Returns: `initiatorID` of type ***String*** - Handle to an initiator element
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.19.29 getBusInterfaceInterfaceMode

Description: Returns the interfaceMode defined on the given busInterface element.

- Returns: `interfaceMode` of type ***String*** - The interface mode (initiator, target, system)
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.19.30 getBusInterfaceMirroredSystemID

Description: Returns the handle to the mirroredSystem element defined on the given busInterface element.

- Returns: `mirroredSystemID` of type ***String*** - Handle to the mirroredSystem element
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.19.31 getBusInterfaceMirroredTargetID

Description: Returns the handle to the mirroredTarget element defined on the given busInterface element.

- Returns: `mirroredTargetID` of type ***String*** - Handle to the mirroredTarget element
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.19.32 getBusInterfaceMonitorID

Description: Returns the handle to the monitor element defined on the given busInterface element.

- Returns: `monitorID` of type ***String*** - Handle to the monitor element
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.19.33 getBusInterfaceRefLocalName

Description: Returns the localName defined on the given busInterfaceRef element.

- Returns: `localName` of type ***String*** - The local name
- Input: `busInterfaceRefID` of type ***String*** - Handle to a busInterfaceRef element

F.7.19.34 getBusInterfaceSystemID

Description: Returns the handle to the system element defined on the given busInterface element.

- Returns: `systemID` of type ***String*** - Handle to the system element
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.19.35 getBusInterfaceTargetID

Description: Returns the handle to the target defined on the given busInterface element.

- Returns: `targetID` of type ***String*** - Handle to the target element
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.19.36 getChannelBusInterfaceRefByID

Description: Returns the busInterface referenced from the given busInterfaceRef element.

- Returns: busInterfaceID of type **String** - Handle to the referenced busInterface element
- Input: busInterfaceRefID of type **String** - Handle to an busInterfaceRef element

F.7.19.37 getChannelBusInterfaceRefIDs

Description: Returns the handles to all the busInterfaceRefs defined on the given channel element.

- Returns: busInterfaceRefIDs of type **String** - List of handles to the busInterfaceRef elements
- Input: channelID of type **String** - Handle to a channel element

F.7.19.38 getFileSetRefGroupGroup

Description: Returns the group defined on the given fileSetRefGroup element.

- Returns: group of type **String** - The group name
- Input: fileSetRefGroupID of type **String** - Handle to a fileSetRefGroup element

F.7.19.39 getIndirectInterfaceBitsInLauID

Description: Returns the handle to the bitsInLau defined on the given indirectInterface element.

- Returns: bitsInLauID of type **String** - Handle to the bitsInLau element
- Input: indirectInterfaceID of type **String** - Handle to an indirectInterface element

F.7.19.40 getInitiatorAddressSpaceRefByName

Description: Returns the addressSpace referenced from the given initiator element.

- Returns: addressSpace of type **String** - The referenced addressSpace name
- Input: initiatorID of type **String** - Handle to a busInterface initiator element

F.7.19.41 getInitiatorAddressSpaceRefID

Description: Returns the addressSpaceRef defined on the given busInterface initiator element.

- Returns: addressSpaceRefID of type **String** - Handle to the addressSpaceRef element
- Input: initiatorID of type **String** - Handle to an initiator element

F.7.19.42 getMemoryMapRefModeRefByID

Description: Returns the handle to the mode referenced from the given memoryMapRef element.

- Returns: modeID of type **String** - Handle to the referenced mode element
- Input: memoryMapRefID of type **String** - Handle to a memoryMapRef element
- Input: modeRef of type **String** - The referenced mode

F.7.19.43 getMemoryMapRefModeRefNames

Description: Returns all the modeRefs defined on the given memoryMapRef element.

- Returns: modeRefs of type **String** - List of the referenced modes
- Input: memoryMapRefID of type **String** - Handle to a memoryMapRef element

F.7.19.44 getMirroredSystemGroup

Description: Returns the group defined on the given mmirroredSystem element

- Returns: `group` of type ***String*** - The group value on the mirroredSystem element
- Input: `mirroredSystemID` of type ***String*** - Handle to a mirroredSystem element

F.7.19.45 getMirroredTargetBaseAddressesID

Description: Returns the handle to baseAddresses defined on the given mirroredTarget element.

- Returns: `baseAddressesID` of type ***String*** - Handle to a baseAddresses element
- Input: `mirroredTargetID` of type ***String*** - Handle to a mirroredTarget element

F.7.19.46 getMonitorGroup

Description: Returns the group defined on the given busInterface monitor element.

- Returns: `group` of type ***String*** - The group name
- Input: `monitorID` of type ***String*** - Handle to a monitor element

F.7.19.47 getRemapAddressModeRefByID

Description: Returns the handle to the mode referenced from the given remapAddress element.

- Returns: `modeID` of type ***String*** - Handle to the referenced mode element
- Input: `remapAddressID` of type ***String*** - Handle to a remapAddress element

F.7.19.48 getRemapAddressesRemapAddressID

Description: Returns the handle to remapAddress from the given remapAddresses element.

- Returns: `remapAddressID` of type ***String*** - Handle to a remapAddress element
- Input: `remapAddressesID` of type ***String*** - Handle to a remapAddresses element

F.7.19.49 getSystemGroup

Description: Returns the group defined on the given busInterface system element.

- Returns: `group` of type ***String*** - The group name
- Input: `systemID` of type ***String*** - Handle to a system element

F.7.19.50 getFileSetRefGroupIDs

Description: Returns the handles to all the fileSetRefgroups defined on the given target element.

- Returns: `fileSetRefgroupIDs` of type ***String*** - List of handles to the fileSetRefGroup elements
- Input: `targetID` of type ***String*** - Handle to a busInterface target element

F.7.19.51 getTargetMemoryMapRefByName

Description: Returns the memoryMapRef defined on the given target element.

- Returns: `memoryMapRef` of type ***String*** - The memoryMap reference
- Input: `targetID` of type ***String*** - Handle to a target element of a busInterface

F.7.19.52 getTargetMemoryMapRefID

Description: Returns the handle to the memoryMapRef defined on the given target element.

- Returns: `memoryMapRefID` of type ***String*** - Handle to the memoryMapRef element
- Input: `targetID` of type ***String*** - Handle to a busInterface target element

F.7.19.53 getTargetTransparentBridgeIDs

Description: Returns the handles to all the transparentBridges defined on the given target element.

- Returns: `transparentBridgesIDs` of type ***String*** - List of handles to the transparentBridge elements
- Input: `targetID` of type ***String*** - Handle to a busInterface target element

F.7.20 Bus interface (EXTENDED)

F.7.20.1 addAbstractionTypePortMap

Description: Adds a portMap with the given name, logicalPortName, and physicalPortName to the given abstractionType

- Returns: `portMapID` of type ***String*** - Handle to a new portMap element
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element
- Input: `logicalPortName` of type ***String*** - Logical port name
- Input: `physicalPortName` of type ***String*** - Physical port name

F.7.20.2 addAbstractionTypeViewRef

Description: Adds a viewRef with the given name to the given abstractionType element.

- Returns: `viewRefID` of type ***String*** - the viewRef identifier
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element
- Input: `viewRef` of type ***String*** - View name

F.7.20.3 addAddressSpaceRefModeRef

Description: Adds an mode reference on an addressSpaceRef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressSpaceRefID` of type ***String*** - Handle to an addressSpaceRef element
- Input: `modeRef` of type ***String*** - The modeRef expression to be added on an addressSpaceRef element

F.7.20.4 addBaseAddressesRemapAddresses

Description: Adds a remapAddress on a baseAddresses element

- Returns: `remapAddressesID` of type ***String*** - Handle to a remapAddresses element
- Input: `baseAddressesID` of type ***String*** - Handle to a baseAddresses element
- Input: `remapAddress` of type ***String*** - The remapAddress value to be set on the remapAddresses element

F.7.20.5 addBusInterfaceAbstractionType

Description: Adds an abstractionType with the given abstractionRef to the given busInterface element.

- Returns: abstractionTypeID of type **String** - Handle to a new abstractionType element
- Input: busInterfaceID of type **String** - Handle to a busInterface element
- Input: abstractionRef of type **String//** - AbstractionDef VLN

F.7.20.6 addChannelBusInterfaceRef

Description: Adds a busInterfaceRef with the given name to the given channel element.

- Returns: busInterfaceRefID of type **String** - Handle to busInterfaceRef element
- Input: channelID of type **String** - Handle to a channel element
- Input: name of type **String** - BusInterface name

F.7.20.7 addMemoryMapRefModeRef

Description: Adds a modeRef on the given memoryMapRef element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryMapRefID of type **String** - Handle to a memoryMapRef element
- Input: modeRef of type **String** - The mode reference to be add on the memoryMapRef element

F.7.20.8 addTargetFileSetRefGroup

Description: Adds a fileSetRefGroup on the given target element

- Returns: fileSetRefGroupID of type **String** - Handle to a fileSetRefGroup element
- Input: targetID of type **String** - Handle to a target element of a busInterface

F.7.20.9 addTargetTransparentBridge

Description: Adds a transparentBridge on the given target element

- Returns: transparentBridgeID of type **String** - Handle of a transparentBrige element
- Input: targetID of type **String** - Handle to a target element of a busInterface
- Input: initiatorRef of type **String** - The initiator reference value to be added

F.7.20.10 removeAbstractionTypePortMap

Description: Removes the given portMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portMapID of type **String** - Handle to a portMap element

F.7.20.11 removeAbstractionTypeViewRef

Description: Removes the viewRef element with the given viewRef ID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewRefID of type **String** - Handle of the viewRef

F.7.20.12 removeAddressSpaceRefBaseAddress

Description: Removes the baseAddress from the given addressSpaceRef element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressSpaceRefID of type **String** - Handle to an addressSpaceRef element

F.7.20.13 removeAddressSpaceRefModeRef

Description: Removes the designated modeRef from the given addressSpaceRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressSpaceRefID of type **String** - Handle to an addressSpaceRef element
- Input: modeRef of type **String** - The mode reference to be removed

F.7.20.14 removeBaseAddressesRemapAddresses

Description: Removes a remapAddresses element from a baseAddresses element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: remapAddressesID of type **String** - Handle to a remapAddresses element

F.7.20.15 removeBusDefinitionSystemGroupName

Description: Removes the given systemGroupName from its containing busDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: systemGroupNameID of type **String** - Handle to the systemGroupName to remove

F.7.20.16 removeBusInterfaceAbstractionType

Description: Removes the given abstractionType element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionTypeID of type **String** - Handle to an abstractionType element

F.7.20.17 removeBusInterfaceBitSteering

Description: Removes bitSteering from the given busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.20.18 removeBusInterfaceBitsInLau

Description: Removes the bitsInLau for the given busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element

F.7.20.19 removeBusInterfaceConnectionRequired

Description: Removes connectionRequired from the given busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.20.20 removeBusInterfaceEndianness

Description: Removes endianness from the given busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to a busInterface element

F.7.20.21 removeChannelBusInterfaceRef

Description: remove the selected busInterfaceRef from the Channel element if busInterfaceRef list size is greater than 2.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceRefID of type **String** - Handle to a busInterfaceRef element

F.7.20.22 removeFileSetRefGroupGroup

Description: Removes the group on the given fileSetRefGroup element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileSetRefGroupID of type **String** - Handle to a fileSetRefGroup element

F.7.20.23 removeIndirectInterfaceBitsInLau

Description: Removes the bitsInLau for the given indirectInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectInterfaceID of type **String** - Handle to an indirectInterface element

F.7.20.24 removeInitiatorAddressSpaceRef

Description: Removes the addressSpaceRef from an initiator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: initiatorID of type **String** - Handle to an initiator element of a busInterface

F.7.20.25 removeMemoryMapRefModeRef

Description: Removes a modeRef from the given memoryMapRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryMapRefID of type **String** - Handle to a memoryMapRef element
- Input: modeRef of type **String** - The mode reference to be remove on the memoryMapRef element

F.7.20.26 removeMirroredTargetBaseAddresses

Description: Removes the baseAddresses from the given mirroredTarget element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: mirroredTargetID of type **String** - Handle to a mirroredTarget element

F.7.20.27 removeMonitorGroup

Description: Removes the group on the given monitor element of a busInterface.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: monitorID of type **String** - Handle to a monitor element of a busInterface

F.7.20.28 removeTargetFileSetRefGroup

Description: Removes a fileSetRefGroup element from the given target element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileSetRefGroupID of type **String** - Handle to a fileSetRefGroup

F.7.20.29 removeTargetMemoryMapRef

Description: Removes the memoryMap reference on a target element of a bus interface.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: targetID of type **String** - Handle to a target element of a busInterface

F.7.20.30 removeTargetTransparentBridge

Description: Removes a transparentBridge element from the given target element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transparentBrigeID of type **String** - Handle to a transparentBrige element

F.7.20.31 setAddressSpaceRefBaseAddress

Description: Sets the baseAddress on an addressSpaceRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressSpaceRefID of type **String** - Handle to an addressSpaceRef element
- Input: baseAddress of type **String** - The baseAddress expression to be set

F.7.20.32 setBaseAddressesRange

Description: Sets the range on a baseAddress element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: baseAddressesID of type **String** - Handle to a baseAddresses element
- Input: range of type **String** - the new range to set

F.7.20.33 setBusInterfaceBitSteering

Description: Sets bitSteering with the given value for the given busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to a busInterface element
- Input: value of type **String** - BitSteering value

F.7.20.34 setBusInterfaceBitsInLau

Description: Sets the given bitsInLau expression for the busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element
- Input: bitsInLau of type **String** - The bitsInLau expression

F.7.20.35 setBusInterfaceBusType

Description: Sets the busType on a busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element
- Input: vlnv of type **String[]** - The vlnv expression to be set on the busType

F.7.20.36 setBusInterfaceConnectionRequired

Description: Sets connectionRequired with the given value for the given busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to a busInterface element
- Input: value of type **Boolean** - ConnectionRequired value

F.7.20.37 setBusInterfaceEndianness

Description: Sets endianness with the given value for the given busInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to a busInterface element
- Input: endianness of type **String** - Endianness value : little or big

F.7.20.38 setBusInterfaceInitiator

Description: Sets the bus interface mode to initiator.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element

F.7.20.39 setBusInterfaceMirroredInitiator

Description: Sets the bus interface mode to mirroredInitiator.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element

F.7.20.40 setBusInterfaceMirroredSystem

Description: Sets the bus interface mode to mirroredSystem.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element
- Input: group of type **String** - The group value to be set on the system element

F.7.20.41 setBusInterfaceMirroredTarget

Description: Sets the bus interface mode to mirroredTarget.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element

F.7.20.42 setBusInterfaceMonitor

Description: Sets the bus interface mode to monitor.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element
- Input: interfaceMode of type **String** - The interfaceMode value to be set on the monitor element

F.7.20.43 setBusInterfaceRefLocalName

Description: Returns the element localName on a busInterfaceRef of a channel element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceRefID of type **String** - Handle to the identifier to a busInterfaceRef element
- Input: name of type **String** - BusInterface name

F.7.20.44 setBusInterfaceSystem

Description: Sets the bus interface mode to system.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element
- Input: group of type **String** - The group value to be set on the system element

F.7.20.45 setBusInterfaceTarget

Description: Sets the bus interface mode to target.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: busInterfaceID of type **String** - Handle to an busInterface element
- Input: memoryMapRef of type **String** - The memoryMap reference set on the target element

F.7.20.46 setFileSetRefGroupGroup

Description: Sets the group on the given fileSetRefGroup element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileSetRefGroupID of type **String** - Handle to a fileSetRefGroup element
- Input: group of type **String** - The group value to be set on the fileSetRefGroup element

F.7.20.47 setFileSetRefLocalName

Description: Sets the localName on a fileSetRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileSetRefID of type **String** - Handle to a fileSetRef element
- Input: localName of type **String** - The local Name value to be set

F.7.20.48 setIndirectInterfaceBitsInLau

Description: Sets the given bitsInLau expression for the indirectInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectInterfaceID of type **String** - Handle to an indirectInterface element
- Input: bitsInLau of type **String** - BitsInLau expression

F.7.20.49 setInitiatorAddressSpaceRef

Description: Sets the addressSpaceRef on an initiator element of a busInterface.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: initiatorID of type **String** - Handle to an initiator element of a busInterface
- Input: addressSpaceRef of type **String** - The addressSpace reference to be set on the initiator element

F.7.20.50 setMirroredSystemGroup

Description: Sets the group on the given mmirroredSystem element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: mirroredSystemID of type **String** - Handle to a mirroredSystem element
- Input: group of type **String** - The group value to be set

F.7.20.51 setMirroredTargetBaseAddresses

Description: Sets the baseAddresses for the given mirroredTarget element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: mirroredTargetID of type **String** - Handle to a mirroredTarget element
- Input: remapAddress of type **String** - The new remapAddress expression
- Input: range of type **String** - The new range expression

F.7.20.52 setMonitorGroup

Description: Sets the group on the given monitor element of a busInterface.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: monitorID of type **String** - Handle to a monitor element of a busInterface
- Input: group of type **String** - The group value to be set

F.7.20.53 setRemapAddressesRemapAddress

Description: Sets the remapAddress from the given remapAddresses element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: remapAddressesID of type **String** - Handle to a remapAddresses element
- Input: remapAddress of type **String** - The remapAddress value to be set

F.7.20.54 setSystemGroup

Description: Sets the group on a system element of a busInterface.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: systemID of type **String** - Handle to a system element of a busInterface
- Input: group of type **String** - The group value to be set

F.7.20.55 setTargetMemoryMapRef

Description: Sets the memoryMap reference on a target element of a bus interface.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: targetID of type **String** - Handle to a target element of a busInterface
- Input: memoryMapRef of type **String** - The memoryMap reference to be set

F.7.21 CPU (BASE)

F.7.21.1 getCpuAddressUnitBits

Description: Returns the addressUnitBits value defined on the given cpu element.

- Returns: addressUnitBits of type **Long** - The addressUnitBits value
- Input: cpuID of type **String** - Handle to a cpu element

F.7.21.2 getCpuAddressUnitBitsExpression

Description: Returns the addressUnitBits expression defined on the given cpu element.

- Returns: addressUnitBits of type **String** - The addressUnitBits expression
- Input: cpuID of type **String** - Handle to a cpu element

F.7.21.3 getCpuAddressUnitBitsID

Description: Returns the handle to the addressUnitBits defined on the given cpu element.

- Returns: addressUnitBitsID of type **String** - Handle to the addressUnitBits element
- Input: cpuID of type **String** - Handle to a cpu element

F.7.21.4 getCpuExecutableImageIDs

Description: Returns the handles to all the executableImages defined on the given cpu element.

- Returns: executableImageIDs of type **String** - List of handles to the executableImage elements
- Input: cpuID of type **String** - Handle to a cpu element

F.7.21.5 getCpuMemoryMapRefByID

Description: Returns the handle to the memoryMap referenced from the given cpu element.

- Returns: memoryMapID of type **String** - Handle to the referenced memoryMap element
- Input: cpuID of type **String** - Handle to a cpu element

F.7.21.6 getCpuMemoryMapRefByName

Description: Returns the memoryMap referenced from the given cpu element.

- Returns: memoryMapRef of type **String** - The referenced memoryMap
- Input: cpuID of type **String** - Handle to a cpu element

F.7.21.7 getCpuRange

Description: Returns the range value on a cpu element.

- Returns: range of type **Long** - The range value
- Input: cpuID of type **String** - Handle to a cpu element

F.7.21.8 getCpuRangeExpression

Description: Returns the range expression defined on the given cpu element.

- Returns: range of type **String** - The range expression
- Input: cpuID of type **String** - Handle to a cpu element

F.7.21.9 getCpuRangeID

Description: Returns the handle to the range defined on the given cpu element.

- Returns: rangeID of type **String** - Handle to the range

- Input: `cpuID` of type ***String*** - Handle to a cpu element

F.7.21.10 getCpuRegionIDs

Description: Returns the handles to all the regions defined on the given cpu element.

- Returns: `regionID` of type ***String*** - List of handles to the region elements
- Input: `cpuID` of type ***String*** - Handle to a cpu element

F.7.21.11 getCpuWidth

Description: Returns the width value defined on the given cpu element.

- Returns: `width` of type ***Long*** - The width value
- Input: `cpuID` of type ***String*** - Handle to a cpu element

F.7.21.12 getCpuWidthExpression

Description: Returns the width expression defined on the given cpu element.

- Returns: `width` of type ***String*** - The width expression
- Input: `cpuID` of type ***String*** - Handle to a cpu element

F.7.21.13 getCpuWidthID

Description: Returns the handle to the width defined on the given cpu element.

- Returns: `widthID` of type ***String*** - Handle to the width element
- Input: `cpuID` of type ***String*** - Handle to a cpu element

F.7.22 CPU (EXTENDED)

F.7.22.1 addCpuExecutableImage

Description: Adds an executable image on a cpu element.

- Returns: `ExecutableImageID` of type ***String*** - the executable image identifier
- Input: `cpuID` of type ***String*** - Handle of a cpu element
- Input: `name` of type ***String*** - The name to set on the executableImage
- Input: `imageID` of type ***String*** - The imageId attribute to set on the executableImage

F.7.22.2 addCpuRegion

Description: Adds a region on a cpu element.

- Returns: `regionID` of type ***String*** - the region identifier
- Input: `cpuID` of type ***String*** - Handle to a cpu element
- Input: `name` of type ***String*** - The name of the region
- Input: `addressOffset` of type ***String*** - The addressOffset expression of the region
- Input: `range` of type ***String*** - The range expression of the region

F.7.22.3 removeCpuAddressUnitBits

Description: Removes an addressUnitBits expression on a cpu element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

- Input: `cpuID` of type ***String*** - Handle to a cpu element

F.7.22.4 removeCpuExecutableImage

Description: Removes the given executableImage from its containing cpu element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.22.5 removeCpuRegion

Description: Removes the given region from its containing cpu element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regionID` of type ***String*** - Handle of a region element

F.7.22.6 setCpuAddressUnitBits

Description: Sets an addressUnitBits expression on a cpu element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `cpuID` of type ***String*** - Handle to a cpu element
- Input: `addressUnitBits` of type ***String*** - The addressunitBits expression to set

F.7.22.7 setCpuMemoryMapRef

Description: Sets the memoryMapRef of the cpu.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `cpuID` of type ***String*** - Handle to a cpu element
- Input: `value` of type ***String*** - Name of the referenced memoryMap

F.7.22.8 setCpuRange

Description: Sets the range on a cpu element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `cpuID` of type ***String*** - Handle to a cpu element
- Input: `range` of type ***String*** - The range value to set

F.7.22.9 setCpuWidth

Description: Sets the width expression on a cpu element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `cpuID` of type ***String*** - Handle to a cpu element
- Input: `width` of type ***String*** - The width expression to set

F.7.22.10 setRegionAddressOffset

Description: Sets the addressOffset expression on a region element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regionID` of type ***String*** - Handle of a region element
- Input: `addressOffset` of type ***String*** - The addressOffset expression

F.7.22.11 setRegionRange

Description: Sets the range expression on a region element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regionID` of type ***String*** - Handle of a region element
- Input: `range` of type ***String*** - the range expression to set

F.7.23 Catalog (BASE)

F.7.23.1 getCatalogAbstractionDefIpxactFileIDs

Description: Returns the handles to all the abstractionDefinitions files defined on the given catalog element.

- Returns: `ipxactFileIDs` of type ***String*** - List of handles to the ipxactFile elements
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.23.2 getCatalogAbstractorsIpxactFileIDs

Description: Returns the handles to all the abstractors files defined on the given catalog element.

- Returns: `ipxactFileIDs` of type ***String*** - List of handles to the ipxactFile elements
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.23.3 getCatalogBusDefinitionsIpxactFileIDs

Description: Returns the handles to all the busDefinitions files defined on the given catalog element.

- Returns: `ipxactFileIDs` of type ***String*** - List of handles to the ipxactFile elements
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.23.4 getCatalogCatalogsIpxactFileIDs

Description: Returns the handles to all the IP-XACT files defined on the given catalog element.

- Returns: `ipxactFileIDs` of type ***String*** - List of handles to the ipxactFile elements
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.23.5 getCatalogComponentsIpxactFileIDs

Description: Returns the handles to all the components files defined on the given catalog element.

- Returns: `ipxactFileIDs` of type ***String*** - List of handles to the ipxactFile elements
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.23.6 getCatalogDesignConfigurationsIpxactFileIDs

Description: Returns the handles to all the designConfigurations files defined on the given catalog element.

- Returns: `ipxactFileIDs` of type ***String*** - List of handles to the ipxactFile elements
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.23.7 getCatalogDesignsIpxactFileIDs

Description: Returns the handles to all the designs files defined on the given catalog element.

- Returns: `ipxactFileIDs` of type ***String*** - List of handles to the ipxactFile elements

- Input: catalogID of type **String** - Handle to a catalog element

F.7.23.8 getCatalogGeneratorChainsIpxactFileIDs

Description: Returns the handles to all the generatorChains files defined on the given catalog element.

- Returns: ipxactFileIDs of type **String** - List of handles to the ipxactFile elements
- Input: catalogID of type **String** - Handle to a catalog element

F.7.23.9 getCatalogTypeDefinitionsIpxactFileIDs

Description: Returns the handles to all the typeDefinitions files defined on the given catalog element.

- Returns: ipxactFileIDs of type **String** - List of handles to the ipxactFile elements
- Input: catalogID of type **String** - Handle to a catalog element

F.7.23.10 getIpxactFileName

Description: Returns the file name of the given IP-XACT element.

- Returns: fileName of type **String** - File name ()
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.23.11 getIpxactFileVInvRefByVLNV

Description: Returns the VLNV defined on the given ipxactFile element.

- Returns: VLNV of type **String** - The VLNV of the referenced IP-XACT file
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24 Catalog (EXTENDED)

F.7.24.1 addCatalogAbstractionDefIpxactFile

Description: Adds ipxactFile with the given VLNV and name to abstractionDefinitions in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: abstractionDefVLNV of type **String[]** - abstractionDef VLNV
- Input: fileName of type **String** - Abstraction definition file name

F.7.24.2 addCatalogAbstractorsIpxactFile

Description: Adds ipxactFile with the given VLNV and name to abstractors in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: abstractorVLNV of type **String[]** - Abstractor VLNV
- Input: fileName of type **String** - Abstractor file name

F.7.24.3 addCatalogBusDefinitionsIpxactFile

Description: Adds ipxactFile with the given VLNV and name to busDefinitions in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: busDefVLNV of type **String[]** - busDef VLNV
- Input: fileName of type **String** - Bus definition file name

F.7.24.4 addCatalogCatalogsIpxactFile

Description: Adds ipxactFile with the given VLNV and name to catalogs in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: catalogVLNV of type **String[]** - Catalog VLNV
- Input: fileName of type **String** - Catalog file name

F.7.24.5 addCatalogComponentsIpxactFile

Description: Adds ipxactFile with the given VLNV and name to components in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: componentVLNV of type **String[]** - Component VLNV
- Input: fileName of type **String** - Component file name

F.7.24.6 addCatalogDesignConfigurationsIpxactFile

Description: Adds ipxactFile with the given VLNV and name to designConfigurations in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: designConfigurationVLNV of type **String[]** - designConfiguration VLNV
- Input: fileName of type **String** - Design configuration file name

F.7.24.7 addCatalogDesignsIpxactFile

Description: Adds ipxactFile with the given VLNV and name to designs in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: designVLNV of type **String[]** - Design VLNV
- Input: fileName of type **String** - Design file name

F.7.24.8 addCatalogGeneratorChainsIpxactFile

Description: Adds ipxactFile with the given VLNV and name to generatorChains in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: generatorChainVLNV of type **String[]** - generatorChain VLNV
- Input: fileName of type **String** - Generator chain file name

F.7.24.9 addCatalogTypeDefinitionsIpxactFile

Description: Adds ipxactFile with the given VLVN and name to typeDefinition in the given catalog element.

- Returns: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: catalogID of type **String** - Handle to a catalog element
- Input: typeDefinitionVLNV of type **String[]** - typeDefinition VLVN
- Input: fileName of type **String** - Generator chain file name

F.7.24.10 removeCatalogAbstractionDefIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.11 removeCatalogAbstractorsIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.12 removeCatalogBusDefinitionsIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.13 removeCatalogCatalogsIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.14 removeCatalogComponentsIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.15 removeCatalogDesignConfigurationsIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.16 removeCatalogDesignsIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.17 removeCatalogGeneratorChainsIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.18 removeCatalogTypeDefinitionsIpxactFile

Description: Removes the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element

F.7.24.19 setIpxactFileName

Description: Sets the name for the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: name of type **String** - Name of the ipxactFile

F.7.24.20 setIpxactFileVInv

Description: Sets the VLVN of the given ipxactFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: ipxactFileID of type **String** - Handle to an ipxactFile element
- Input: vlnv of type **String[]** - VLVN of the ipxactFile

F.7.25 Choice (BASE)

F.7.25.1 getChoiceEnumerationIDs

Description: Returns the handles to all the choiceEnumerations defined on the given choice element.

- Returns: choiceEnumerationIDs of type **String** - List of handles to choiceEnumeration elements
- Input: choiceID of type **String** - Handle to a choice element

F.7.25.2 getEnumerationValue

Description: Returns the enumerationValue defined on the given enumeration element.

- Returns: enumerationValue of type **String** - The enumeration value
- Input: choiceEnumerationID of type **String** - Handle to an enumeration element

F.7.25.3 getEnumerationValueExpression

Description: Returns the expression defined on the given enumeration element.

- Returns: enumerationExpression of type **String** - The enumeration expression
- Input: choiceEnumerationID of type **String** - Handle to an enumeration element

F.7.26 Choice (EXTENDED)

F.7.26.1 addChoiceEnumeration

Description: Adds enumeration with the given name to the given choice element.

- Returns: choiceEnumerationID of type **String** - Handle to a new choiceEnumeration
- Input: choiceID of type **String** - Handle to a choice element
- Input: name of type **String** - ChoiceEnumeration name

F.7.26.2 removeChoiceEnumeration

Description: Removes the given choiceEnumeration.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: choiceEnumerationID of type **String** - Handle to a choiceEnumeration element

F.7.26.3 setEnumerationValue

Description: Sets the value on the given enumeration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: choiceEnumerationID of type **String** - Handle to the identifier of an enumeration element
- Input: value of type **String** - New value expression

F.7.27 Clearbox (BASE)

F.7.27.1 getClearboxElementClearboxType

Description: Returns the clearboxType element defined on the given clearboxElement.

- Returns: type of type **String** - The clearboxType value
- Input: clearboxElementID of type **String** - Handle to a clearboxElement element

F.7.27.2 getClearboxElementDriveable

Description: Returns the driveable element defined on the given clearboxElement.

- Returns: value of type **Boolean** - The driveable value
- Input: clearboxElementID of type **String** - Handle to a clearboxElement

F.7.27.3 getClearboxElementRefByID

Description: Returns the handle to the clearboxElement referenced from the name define on given clearboxElementRef element.

- Returns: clearboxElementID of type **String** - Handle to the referenced clearboxElement
- Input: clearboxElementRefID of type **String** - Handle to a clearboxElementRef element

F.7.27.4 getClearboxElementRefLocationIDs

Description: Returns the handles to all the clearboxElementRefLocations defined on the given clearboxElementRef element.

- Returns: `clearboxElementRefLocationIDs` of type ***String*** - List of handles to the `clearboxElementRefLocation` elements
- Input: `clearboxElementRefID` of type ***String*** - Handle to a `clearboxElementRef` element

F.7.28 Clearbox (EXTENDED)

F.7.28.1 addClearboxElementRefLocation

Description: Adds a `clearboxElementRefLocation` element to a given `clearboxElementRef` element.

- Returns: `clearboxElementRefLocationID` of type ***String*** - Handle to the added `clearboxElementRefLocation`
- Input: `clearboxElementRefID` of type ***String*** - Handle to a `clearboxElementRef` element
- Input: `value` of type ***String*** - value of the `pathSegment`

F.7.28.2 addLocationSlice

Description: Adds a slice to a given `Location` element.

- Returns: `sliceID` of type ***String*** - Handle to the added slice
- Input: `locationID` of type ***String*** - Handle to a `location` element
- Input: `value` of type ***String*** - Handle to value of the `pathSegment`

F.7.28.3 removeClearboxElementDriveable

Description: Removes driveable from the given `clearBox` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `clearboxElementID` of type ***String*** - Handle to a `clearboxElement` element

F.7.28.4 removeClearboxElementRefLocation

Description: Removes the given `clearboxElementRefLocation` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `clearboxElementRefLocationID` of type ***String*** - Handle to a `clearboxElementRefLocation` element

F.7.28.5 setClearboxElementClearboxType

Description: Sets clearbox type.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `clearboxID` of type ***String*** - Handle to a clearbox element
- Input: `clearboxType` of type ***String*** - clearbox type

F.7.28.6 setClearboxElementDriveable

Description: Sets driveable with the given value for the given `clearBox` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `clearboxElementID` of type ***String*** - Handle to a `clearboxElement` element
- Input: `value` of type ***Boolean*** - ClearboxElement driveable

F.7.29 Component (BASE)

F.7.29.1 getComponentAddressSpaceIDs

Description: Returns the addressSpaceIDs defined on the given component object or component instance element.

- Returns: addressSpaceIDs of type **String** - List of handles to the addressSpace elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.2 getComponentBusInterfaceIDs

Description: Returns the handles to all the busInterfaces defined on the given component object or component instance element.

- Returns: busInterfaceIDs of type **String** - List of handles to the busInterface elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.3 getComponentChannelIDs

Description: Returns the channelIDs defined on the given component object or component instance element.

- Returns: channelIDs of type **String** - List of handles to the channel elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.4 getComponentChoiceIDs

Description: Returns the handles to all the choices defined on the given component object or component instance element.

- Returns: choiceIDs of type **String** - List of handles to choice elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.5 getComponentClearboxElementIDs

Description: Returns the handles to all the clearboxElements defined on a given component or component instance element.

- Returns: clearboxElementIDs of type **String** - List of handles to the clearboxElement elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component or componentInstance element

F.7.29.6 getComponentComponentGeneratorIDs

Description: Returns the handles to all the generators defined on the given component or component instance element.

- Returns: generatorIDs of type **String** - List of handles to the generator elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component or componentInstance element

F.7.29.7 getComponentComponentInstantiationIDs

Description: Returns the handles to all the componentInstantiations defined on the given component object or component instance element.

- Returns: componentInstantiationIDs of type **String** - List of handles to the componentInstantiation elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.8 getComponentCpuIDs

Description: Returns the handles to all the cpus defined on the given component object or component instance element.

- Returns: cpuIDs of type **String** - List of handles to the cpu elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.9 getComponentDesignConfigurationInstantiationIDs

Description: Returns the handles to all the designConfigurationInstantiations defined on the given component or component instance element.

- Returns: designConfigurationInstantiationIDs of type **String** - List of handles to the designConfigurationInstantiation elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.10 getComponentDesignInstantiationIDs

Description: Returns the handles to all the designInstantiations defined on the given component object or component instance element.

- Returns: designInstantiationIDs of type **String** - List of handles to the designInstantiation elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.11 getComponentExternalTypeDefinitionsIDs

Description: Returns the handles to all the typeDefinitions defined on the given component object.

- Returns: externalTypeDefinitionsIDs of type **String** - List of handles to the typeDefinition elements
- Input: componentID of type **String** - Handle to a component object

F.7.29.12 getComponentFileSetIDs

Description: Returns the handles to all the fileSets defined on the given component object or component instance element.

- Returns: fileSetIDs of type **String** - List of handles to fileSet elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.13 getComponentIndirectInterfaceIDs

Description: Returns the indirectInterfaceIDs defined on the given component object or component instance element.

- Returns: `indirectInterfaceIDs` of type ***String*** - List of handles to the indirectInterface elements
- Input: `componentOrComponentInstanceID` of type ***String*** - Handle to a component object or componentInstance element

F.7.29.14 getComponentMemoryMapIDs

Description: Returns the memoryMapIDs defined on the given component object or component instance element.

- Returns: `memoryMapIDs` of type ***String*** - List of handles to the memoryMap elements
- Input: `componentOrComponentInstanceID` of type ***String*** - Handle to a component object or componentInstance element

F.7.29.15 getComponentModelIDs

Description: Returns the handles to all the modes defined on the given component object

- Returns: `modeIDs` of type ***String*** - List of handles to the mode elements
- Input: `componentID` of type ***String*** - Handle to a component object

F.7.29.16 getComponentOtherClockDriverIDs

Description: Returns the handles to all the clockDrivers defined on the given component object or component instance element.

- Returns: `clockDriverIDs` of type ***String*** - List of handles to the clockDriver elements
- Input: `componentOrComponentInstanceID` of type ***String*** - Handle to a component object or componentInstance element

F.7.29.17 getComponentPortIDs

Description: Returns the handles to all the ports defined on the given component object or component instance element.

- Returns: `portIDs` of type ***String*** - List of handles to port elements
- Input: `componentOrComponentInstanceID` of type ***String*** - Handle to a component object or componentInstance element

F.7.29.18 getComponentPowerDomainIDs

Description: Returns the handles to all the powerDomains defined on the given component object.

- Returns: `powerDomainIDs` of type ***String*** - List of handles to the powerDomain elements
- Input: `componentID` of type ***String*** - Handle to a component object

F.7.29.19 getComponentResetTypeIDs

Description: Returns the handles to all the resetTypes defined on the given component object or component instance element.

- Returns: `resetTypeID`s of type ***String*** - List of handles to the resetType elements

- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.20 getComponentSelectedViewIDs

Description: Returns the handles to the selected views defined on the given component object or component instance element.

- Returns: viewIDs of type **String** - List of handles to the selected view elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.21 getComponentViewIDs

Description: Returns the viewIDs defined on the given component object or component instance element.

- Returns: viewIDs of type **String** - List of handles to the view elements
- Input: componentOrComponentInstanceID of type **String** - Handle to a component object or componentInstance element

F.7.29.22 getModeCondition

Description: Returns the condition (resolved) value defined on the given component mode element.

- Returns: condition of type **String** - The mode condition value
- Input: modeID of type **String** - Handle to a mode element

F.7.29.23 getModeConditionExpression

Description: Returns the condition expression defined on the given component mode element.

- Returns: condition of type **String** - The mode condition expression
- Input: modeID of type **String** - Handle to a mode element

F.7.29.24 getModeName

Description: Returns the name of the given component mode element.

- Returns: name of type **String** - The mode name
- Input: modeID of type **String** - Handle to a mode element

F.7.30 Component (EXTENDED)

F.7.30.1 addComponentAddressSpace

Description: Adds an addressSpace with the given name, range, and width to the given component element.

- Returns: addressSpaceID of type **String** - Handle to a new addressSpace
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - addressSpace name
- Input: range of type **String** - addressSpace range expression
- Input: width of type **String** - addressSpace width expression

F.7.30.2 addComponentChannel

Description: Adds a channel with the given name and busInterfaceRefs to the given component element.

- Returns: channelID of type **String** - Handle to a new channel
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - Channel name
- Input: busInterfaceRefs of type **String[]** - List of busInterface names

F.7.30.3 addComponentChoice

Description: Adds a choice with the given name and enumerations to the given component element.

- Returns: choiceID of type **String** - Handle to a new choice
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - Choice name
- Input: enumerations of type **String[]** - List of enumeration values

F.7.30.4 addComponentClearboxElement

Description: Adds a clearboxElement with the given name and type to the given component element.

- Returns: clearboxElementID of type **String** - Handle to a new clearboxElement
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - ClearboxElement name
- Input: type of type **String** - Clearbox element type

F.7.30.5 addComponentComponentGenerator

Description: Adds a generator with the given name and path to the given component element.

- Returns: generatorID of type **String** - Handle to a new generator
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - Generator name
- Input: generatorExecutable of type **String** - Path to generator executable

F.7.30.6 addComponentComponentInstantiation

Description: Adds a componentInstantiation with the given name to the given component element.

- Returns: componentInstantiationID of type **String** - Handle to a new componentInstantiation
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - ComponentInstantiation name

F.7.30.7 addComponentCpu

Description: Adds a cpu with the given name to the given element.

- Returns: cpuID of type **String** - Handle to the added cpu
- Input: componentID of type **String** - Handle to a component object
- Input: name of type **String** - The cpu name
- Input: memoryMapRef of type **String** - A ref to an memoryMap element
- Input: range of type **String** - The range expression to set

- Input: width of type **String** - The width expression to set

F.7.30.8 addComponentDesignConfigurationInstantiation

Description: Adds a designConfiguration instantiation element to the given component or componentInstance element.

- Returns: designConfigurationInstantiationID of type **String** - Handle of new designConfigurationInstantiation
- Input: componentOrComponentInstanceID of type **String** - Handle to a component element
- Input: name of type **String** - designConfigurationInstantiation name
- Input: designConfigurationVLNV of type **String[]** - VLNV for a new designConfigurationInstantiation

F.7.30.9 addComponentDesignInstantiation

Description: Adds a designInstantiation with the given name and designRef to the given component element.

- Returns: designInstantiationID of type **String** - Handle to a new designInstantiation
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - DesignInstantiation name
- Input: designVLNV of type **String[]** - Design VLNV

F.7.30.10 addComponentFileSet

Description: Adds a fileSet with the given name to the given component element.

- Returns: fileSetID of type **String** - Handle to a new fileSet
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - FileSet name

F.7.30.11 addComponentIndirectInterface

Description: Adds an indirectInterface with the given name, indirectAddress, indirectData, memoryMapRef, or busInterfaceRef to the given component element.

- Returns: indirectInterfaceID of type **String** - Handle to a new indirectInterface
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - The indirectInterface name
- Input: indirectAddressRef of type **String** - Set the fieldRef on the indirectAddressRef
- Input: indirectData of type **String** - The indirectData value
- Input: memoryMapRef of type **String** - Name of memoryMap or null

F.7.30.12 addComponentInitiatorBusInterface

Description: Adds a busInterface with the given name, busType, interfaceMode, and group to the given component element.

- Returns: busInterfaceID of type **String** - Handle to a new busInterface
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - The busInterface name
- Input: busVLNV of type **String[]** - The busDef VLNV

F.7.30.13 addComponentMemoryMap

Description: Adds a memoryMap with the given name to the given component element.

- Returns: `memoryMapID` of type ***String*** - Handle to a new memoryMap
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: name of type ***String*** - MemoryMap name

F.7.30.14 addComponentMirroredInitiatorBusInterface

Description: Adds a busInterface with the given name, busType, interfaceMode, and group to the given component element.

- Returns: `busInterfaceID` of type ***String*** - Handle to a new busInterface
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: name of type ***String*** - The busInterface name
- Input: `busVLAN` of type ***String[]*** - The busDef VLVN

F.7.30.15 addComponentMirroredSystemBusInterface

Description: Adds a busInterface with the given name, busType, interfaceMode, and group to the given component element.

- Returns: `busInterfaceID` of type ***String*** - Handle to a new busInterface
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: name of type ***String*** - The busInterface name
- Input: `busVLAN` of type ***String[]*** - The busDef VLVN
- Input: group of type ***String*** - Sets the group

F.7.30.16 addComponentMirroredTargetBusInterface

Description: Adds a busInterface with the given name, busType, interfaceMode, and group to the given component element.

- Returns: `busInterfaceID` of type ***String*** - Handle to a new busInterface
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: name of type ***String*** - The busInterface name
- Input: `busVLAN` of type ***String[]*** - The busDef VLVN

F.7.30.17 addComponentMode

Description: Adds a new mode to the given component.

- Returns: `modeID` of type ***String*** - The identifier of the added mode
- Input: `componentID` of type ***String*** - Handle to the component object
- Input: name of type ***String*** - Mode name

F.7.30.18 addComponentMonitorBusInterface

Description: Adds a busInterface with the given name, busType, interfaceMode, and group to the given component element.

- Returns: `busInterfaceID` of type ***String*** - Handle to a new busInterface
- Input: `componentID` of type ***String*** - Handle to a component element

- Input: name of type **String** - busInterface name
- Input: busVNV of type **String[]** - busDef VNV
- Input: interfaceMode of type **String** - InterfaceMode attribute. Can be one of 'initiator', 'target', 'system', 'mirroredInitiator', 'mirroredTarget' or 'mirroredSystem'

F.7.30.19 addComponentOtherClockDriver

Description: Adds a clockDriver with the given name, period, offset, value, and duration to the given component element.

- Returns: clockDriverID of type **String** - Handle to a new clockDriver
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - clockDriver name
- Input: periodExpression of type **String** - Clock period
- Input: offsetExpression of type **String** - Clock pulse offset
- Input: valueExpression of type **String** - Clock pulse value
- Input: durationExpression of type **String** - Clock pulse duration

F.7.30.20 addComponentResetType

Description: Adds a resetType with the given name to the given component element.

- Returns: resetTypeID of type **String** - Handle to a new resetType
- Input: componentID of type **String** - Handle to a component element
- Input: name of type **String** - resetType name

F.7.30.21 addComponentStructuredInterfacePort

Description: Adds a structured interface port with the given name and direction for the given component element.

- Returns: portID of type **String** - Handle to a new port
- Input: componentID of type **String** - Handle to an component element
- Input: name of type **String** - Handle to a port by his name
- Input: structPortTypeDefTypeName of type **String** - The typeName of the structPortTypeDef

F.7.30.22 addComponentStructuredStructPort

Description: Adds a structured struct port with the given name and direction for the given component element.

- Returns: portID of type **String** - Handle to a new port
- Input: componentID of type **String** - Handle to an component element
- Input: name of type **String** - Handle to a port by his name
- Input: structPortTypeDefTypeName of type **String** - The typeName of the structPortTypeDef
- Input: direction of type **String** - Value of the direction

F.7.30.23 addComponentStructuredUnionPort

Description: Adds a structured union port with the given name and direction for the given component element.

- Returns: `portID` of type ***String*** - Handle to a new port
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: name of type ***String*** - Handle to a port by his name
- Input: `structPortTypeDefType` of type ***String*** - The `typeName` of the `structPortTypeDef`
- Input: `subPortName` of type ***String*** - The name of the subPort
- Input: `direction` of type ***String*** - Value of the direction

F.7.30.24 addComponentSystemBusInterface

Description: Adds a busInterface with the given name, busType, interfaceMode, and group to the given component element.

- Returns: `busInterfaceID` of type ***String*** - Handle to a new busInterface
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: name of type ***String*** - busInterface name
- Input: `busVLAN` of type ***String*** - busDef VLAN
- Input: group of type ***String*** - group name

F.7.30.25 addComponentTargetBusInterface

Description: Adds a busInterface with the given name, busType, interfaceMode, and group to the given component element.

- Returns: `busInterfaceID` of type ***String*** - Handle to a new busInterface
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: name of type ***String*** - busInterface name
- Input: `busVLAN` of type ***String*** - busDef VLAN

F.7.30.26 addComponentTransactionalPort

Description: Adds a transactional port with the given name and initiative to the given component element.

- Returns: `portID` of type ***String*** - Handle to a new port
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: name of type ***String*** - Port name
- Input: initiative of type ***String*** - Port initiative

F.7.30.27 addComponentView

Description: Adds a view with the given name to the given component object.

- Returns: `viewID` of type ***String*** - Handle to the added view
- Input: `componentID` of type ***String*** - Handle to a component object
- Input: name of type ***String*** - View name

F.7.30.28 addComponentWirePort

Description: Adds a wire port with the given name and direction to the given component element.

- Returns: `portID` of type ***String*** - Handle to a new port
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - Port name
- Input: `direction` of type ***String*** - Port direction

F.7.30.29 removeComponentAddressSpace

Description: Removes the given addressSpace element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.30.30 removeComponentBusInterface

Description: Removes the given busInterface element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.30.31 removeComponentChannel

Description: Removes the given channel element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `channelID` of type ***String*** - Handle to a channel element

F.7.30.32 removeComponentChoice

Description: Removes the given choice element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `choiceID` of type ***String*** - Handle to a choice element

F.7.30.33 removeComponentClearboxElement

Description: Removes the given clearboxElement element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `clearboxElementID` of type ***String*** - Handle to a clearboxElement element

F.7.30.34 removeComponentComponentGenerator

Description: Removes the given generator element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.30.35 removeComponentComponentInstantiation

Description: Removes the given componentInstantiation element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.30.36 removeComponentCpu

Description: Removes the given cpu element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: cpuID of type **String** - Handle to a cpu

F.7.30.37 removeComponentDesignConfigurationInstantiation

Description: Removes the given designConfigurationInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: designConfigurationInstantiationID of type **String** - Handle to a designConfigurationInstantiation element

F.7.30.38 removeComponentDesignInstantiation

Description: Removes the given designInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: designInstantiationID of type **String** - Handle to a designInstantiation element

F.7.30.39 removeComponentFileSet

Description: Removes the given fileSet element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileSetID of type **String** - Handle to a fileSet element

F.7.30.40 removeComponentIndirectInterface

Description: Removes the given indirectInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectInterfaceID of type **String** - Handle to an indirectInterface element

F.7.30.41 removeComponentMemoryMap

Description: Removes the given memoryMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.30.42 removeComponentMode

Description: Removes the given mode from its containing component object.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeID of type **String** - Handle to the modeID

F.7.30.43 removeComponentOtherClockDriver

Description: Removes the given clockDriver element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: `clockDriverID` of type ***String*** - Handle to a `clockDriver` element

F.7.30.44 removeComponentPort

Description: Removes the given port element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a `port` element

F.7.30.45 removeComponentResetType

Description: Removes the given resetType element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `resetTypeID` of type ***String*** - Handle to a `resetType` element

F.7.30.46 removeComponentView

Description: Removes the given view element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `viewID` of type ***String*** - Handle to a `view` element

F.7.30.47 removeMemoryMapAddressUnitBits

Description: Removes the addressUnitBits on a `memoryMap` or `memoryMapDefinition` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `memoryMapID` of type ***String*** - the `memoryMap` or `memoryMapDefinition` identifier

F.7.30.48 removeModeCondition

Description: Removes the condition associated with the given mode on a component.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `modeID` of type ***String*** - Handle to the mode type

F.7.30.49 removePowerDomainAlwaysOn

Description: Removes alwaysOn from a `powerDomain` element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `powerDomainID` of type ***String*** - Handle to a `powerDomain` element

F.7.30.50 setIndirectInterfaceIndirectAddressRef

Description: set an `indirectAddressRef` node on an `indirectInterface` element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indirectInterfaceID` of type ***String*** - Handle to an `indirectInterface` element
- Input: `fieldRef` of type ***String*** - Set the `fieldRef` on the `indirectAddressRef` element

F.7.30.51 setIndirectInterfaceIndirectDataRef

Description: set an `indirectDataRef` node on an `indirectInterface` element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

- Input: indirectInterfaceID of type **String** - Handle to an indirectInterface element
- Input: fieldRef of type **String** - Set the fieldRef on the indirectAddressRef element

F.7.30.52 setMemoryMapAddressUnitBits

Description: Sets the addressUnitBits on a memoryMap or memoryMapDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryMapID of type **String** - the memoryMap or memoryMapDefinition identifier
- Input: addressUnitBit of type **String** - the addressUnitBit value to be added

F.7.30.53 setModeCondition

Description: Sets the name associated with the given mode on a component.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeID of type **String** - Handle to the mode type
- Input: condition of type **String** - Handle to value of the condition of type String

F.7.30.54 setModeName

Description: Sets the name associated with the given mode on a component.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeID of type **String** - Handle to the mode type
- Input: name of type **String** - Handle to value of the name of type String

F.7.31 Configurable element (BASE)

F.7.31.1 getConfigurableElementIDs

Description: Returns the handles to all the configurableElements defined on the given unconfigured element.

- Returns: configurableElementIDs of type **String** - List of handles to configurable elements
- Input: unconfiguredElementID of type **String** - Handle to an unconfigured element

F.7.31.2 getConfigurableElementValue

Description: Returns the default value defined on the given configurable element.

- Returns: value of type **String** - The default configurable element value
- Input: configurableElementID of type **String** - Handle to a configurable element

F.7.31.3 getConfigurableElementValueExpression

Description: Returns the default expression defined on the given configurable element.

- Returns: expression of type **String** - The default configurable element expression
- Input: configurableElementID of type **String** - Handle to a configurable element

F.7.31.4 getConfigurableElementValueIDs

Description: Returns the handles to all the configurableElementValues defined on the given configured element.

- Returns: configurableElementValueIDs of type **String** - List of handles to configurableElementValue elements
- Input: configuredElementID of type **String** - Handle to a configured element

F.7.31.5 getConfigurableElementValueReferenceID

Description: Returns the referenceId attribute defined on the given configurable element value.

- Returns: referenceID of type **String** - The referenceId value
- Input: configurableElementValueID of type **String** - Handle to a configurable element value

F.7.31.6 getConfigurableElementValueValueExpression

Description: Returns the expression defined on the given configurable element value.

- Returns: expression of type **String** - The expression defined on the configurable element value
- Input: configurableElementValueID of type **String** - Handle to a configurable element value

F.7.31.7 getUnconfiguredID

Description: Returns the handle to the unconfigured element corresponding to the given configured element.

- Returns: unconfiguredElementID of type **String** - Handle to the corresponding unconfigured element
- Input: configuredElementID of type **String** - Handle to a configured element

F.7.32 Configurable element (EXTENDED)

F.7.32.1 addConfigurableElementValue

Description: Adds a configurable element value with the given expression to the given configured element and the given referenceID.

- Returns: configurableElementValueID of type **String** - Handle to new configurableElementValue
- Input: configuredElementID of type **String** - Handle to a configured element
- Input: referenceID of type **String** - Reference to a configurable element
- Input: expression of type **String** - New expression

F.7.32.2 addViewConfigurationConfigurableElementValue

Description: Adds a configurable element value with the given expression to the given configured element and the given referenceID.

- Returns: configurableElementValueID of type **String** - Handle to new configurableElementValue
- Input: viewConfigurationID of type **String** - Handle of a viewConfiguration
- Input: referenceID of type **String** - Reference to a configurable element
- Input: expression of type **String** - New expression

F.7.32.3 removeConfigurableElementValue

Description: Removes the given configurable element value.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: configurableElementValueID of type **String** - Handle to a configurable element value

F.7.32.4 setConfigurableElementValue

Description: Sets the value of a given configurable element (default value).

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: configurableElementID of type **String** - Handle to a configurable element
- Input: expression of type **String** - New expression

F.7.32.5 setConfigurableElementValueReferenceID

Description: Sets the given referenceID for the given configurable element value.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: configurableElementValueID of type **String** - Handle to a configurable element value
- Input: referenceID of type **String** - New referenceID

F.7.32.6 setConfigurableElementValueValue

Description: Sets the given expression as value for the given configurable element value.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: configurableElementValueID of type **String** - Handle to a configurable element value
- Input: expression of type **String** - New expression

F.7.33 Constraint (BASE)

F.7.33.1 getCellSpecificationCellClass

Description: Returns the cellClass value defined on the given cellSpecification element.

- Returns: cellClassID of type **String** - The cellClass value
- Input: cellSpecificationID of type **String** - Handle to a cellSpecification element

F.7.33.2 getCellSpecificationCellFunction

Description: Returns the cellFunction value defined on the given cellSpecification element.

- Returns: cellFunction of type **String** - The cellFunction value
- Input: cellSpecificationID of type **String** - Handle to a cellSpecification element

F.7.33.3 getCellSpecificationCellFunctionID

Description: Returns the handle to the cellFunction defined on the given cellSpecification element.

- Returns: cellFunctionID of type **String** - Handle to the cellFunction element
- Input: cellSpecificationID of type **String** - Handle to a cellSpecification element

F.7.33.4 getConstraintSetDriveConstraintCellSpecificationID

- Description: Returns the handle to the cellSpecification defined on the given constraintSet element.
- Returns: `cellSpecificationID` of type ***String*** - Handle to the cellSpecification element
 - Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.33.5 getConstraintSetLoadConstraintID

- Description: Returns the handle to the loadConstraint defined on the given constraintSet element.
- Returns: `loadConstraintID` of type ***String*** - Handle to the loadConstraint element
 - Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.33.6 getConstraintSetRefLocalName

- Description: Returns the localName defined on the given constraintSetRef element.
- Returns: `localName` of type ***String*** - The local name
 - Input: `constraintSetRefID` of type ***String*** - Handle to a constraintSetRef element

F.7.33.7 getConstraintSetReferenceName

- Description: Returns the value of the constraintSetId attribute defined on the given constraintSet element.
- Returns: `referenceName` of type ***String*** - The constraintSetId attribute value
 - Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.33.8 getConstraintSetTimingConstraintIDs

- Description: Returns the handles to all the timingConstraints defined on the given constraintSet.
- Returns: `timingConstraintIDs` of type ***String*** - List of handles to the timingConstraint elements
 - Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.33.9 getConstraintSetVector

- Description: Returns the range defined on the given constraintSet.
- Returns: `range` of type ***Long*** - Array of two range values: left and right
 - Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.33.10 getConstraintSetVectorExpression

- Description: Returns the vector left and right expressions defined on the given constraintSet element.
- Returns: `range` of type ***String*** - Array of two vector expressions: left and right
 - Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.33.11 getConstraintSetVectorLeftID

- Description: Returns the handle to the left side of the vector defined on the given constraintSet element.
- Returns: `leftID` of type ***String*** - Handle to the left element
 - Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.33.12 getConstraintSetVectorRightID

Description: Returns the handle to the right side of the vector defined on the given constraintSet element.

- Returns: `rightID` of type ***String*** - Handle to the right element
- Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.33.13 getDriveConstraintOther

Description: Returns the other attribute value defined on the given driveConstraint (cellFunction only).

- Returns: `value` of type ***String*** - The value of the other attribute
- Input: `driveConstraintID` of type ***String*** - Handle to a driveConstraint element

F.7.33.14 getDriveConstraintType

Description: Returns the type defined on the given driveConstraint element.

- Returns: `value` of type ***String*** - The drive constraint type
- Input: `driveConstraintID` of type ***String*** - Handle to a driveConstraint element

F.7.33.15 getDriveConstraintValue

Description: Returns the value defined on the given driveConstraint element.

- Returns: `value` of type ***String*** - The drive constraint value
- Input: `driveConstraintID` of type ***String*** - Handle to a driveConstraint element

F.7.33.16 getLoadConstraintCellSpecificationID

Description: Returns the handle to the cellSpecification defined on the given loadConstraint element.

- Returns: `cellSpecificationID` of type ***String*** - Handle to the cellSpecification element
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element

F.7.33.17 getLoadConstraintCount

Description: Returns the count defined on the given loadConstraint element.

- Returns: `value` of type ***Long*** - The load constraint count
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element

F.7.33.18 getLoadConstraintCountExpression

Description: Returns the count expression defined on the given loadConstraint element.

- Returns: `expression` of type ***String*** - The load constraint count
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element

F.7.33.19 getLoadConstraintCountID

Description: Returns the handle to the count element defined on the given loadConstraint element.

- Returns: `countID` of type ***String*** - Handle to the count element
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element

F.7.33.20 getLoadConstraintOther

Description: Returns the other attribute value defined on the given loadConstraint (cellFunction only).

- Returns: value of type **String** - The value of the other attribute
- Input: loadConstraintID of type **String** - Handle to a loadConstraint element

F.7.33.21 getLoadConstraintType

Description: Returns the type defined on the given loadConstraint element.

- Returns: value of type **String** - The load constraint type
- Input: loadConstraintID of type **String** - Handle to a loadConstraint element

F.7.33.22 getLoadConstraintValue

Description: Returns the value defined on the given loadConstraint element.

- Returns: value of type **String** - The load constraint value
- Input: loadConstraintID of type **String** - Handle to a loadConstraint element

F.7.33.23 getTimingConstraintValue

Description: Returns the value defined on the given timingConstraint element.

- Returns: value of type **Float** - The constraint value (in cycle time percentage)
- Input: timingConstraintID of type **String** - Handle to a timingConstraint element

F.7.34 Constraint (EXTENDED)

F.7.34.1 removeConstraintSetDriveConstraint

Description: Removes the given driveConstraint.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetID of type **String** - Handle to a constraintSet element

F.7.34.2 removeConstraintSetLoadConstraint

Description: Removes the given loadConstraint.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetID of type **String** - Handle to a constraintSet element

F.7.34.3 removeConstraintSetTimingConstraint

Description: Removes the given timingConstraint.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: timingConstraintID of type **String** - Handle to a timingConstraint element

F.7.34.4 removeConstraintSetVector

Description: Removes vector from the given constraintSet.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetID of type **String** - Handle to a constraintSet element

F.7.34.5 removeLoadConstraintCount

Description: Removes the given driveConstraint element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: loadConstraintID of type **String** - Handle to a loadConstraint element

F.7.34.6 setCellSpecificationCellClass

Description: Sets the cellClass value of the CellSpecification element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: cellSpecificationID of type **String** - Handle to a cellSpecification element
- Input: cellClass of type **String** - Enumerated value. Can be one of: combinational or sequential

F.7.34.7 setCellSpecificationCellFunction

Description: Sets the cellFunction value of the CellSpecification element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: cellSpecificationID of type **String** - Handle to a cellSpecification element
- Input: cellFunction of type **String** - Enumerated value. Can be of of: nand2, buf, inv, mux21, dff, latch, xor2, or other

F.7.34.8 setConstraintSetDriveConstraint

Description: Sets a new driveConstraint with the given type to the given constraintSet.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetID of type **String** - Handle to a constraintSet element
- Input: cellFunctionOrCellClass of type **String** - The driveConstraint type. Can be : nand2, buf, inv, mux21, dff, latch, xor2, other, combinational or sequential

F.7.34.9 setConstraintSetLoadConstraint

Description: Adds a new loadConstraint with the given type to the given constraintSet.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetID of type **String** - Handle to a constraintSet element
- Input: cellFuntionOrCellClass of type **String** - The loadConstraint type

F.7.34.10 setConstraintSetReferenceName

Description: Sets the given referenceName for the constraintSetId attribute for the given constraint set.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetID of type **String** - Handle to a constraintSet element
- Input: referenceName of type **String** - The constraintSetId attribute value

F.7.34.11 setConstraintSetVector

Description: Sets the constraintSet vector left and right values.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetID of type **String** - Handle to a constraintSet element
- Input: vector of type **String[]** - Vector left expression and right expression

F.7.34.12 setDriveConstraintOtherValue

Description: Sets the other attribute value for the given driveConstraint (cellFunction only).

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: driveConstraintID of type **String** - Handle to a driveConstraint element
- Input: other of type **String** - Value of the other attribute

F.7.34.13 setDriveConstraintValue

Description: Sets the given value for the given driveConstraint element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: driveConstraintID of type **String** - Handle to a driveConstraint element
- Input: value of type **String** - The driveConstraint value

F.7.34.14 setLoadConstraintCellSpecification

Description: Sets the cellSpecification on the given loadConstraint element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: loadConstraintID of type **String** - Handle to a loadConstraintElement
- Input: cellFunctionOrCellClass of type **String** - Define the type of the cellSpecification depending on the value

F.7.34.15 setLoadConstraintCount

Description: Sets the given count for the given loadConstraint element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: loadConstraintID of type **String** - Handle to a loadConstraint element
- Input: count of type **String** - The loadConstraint count

F.7.34.16 setLoadConstraintOtherValue

Description: Sets the other attribute value for the given loadConstraint (cellFunction only).

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: loadConstraintID of type **String** - Handle to a loadConstraint element
- Input: other of type **String** - Value of the other attribute

F.7.34.17 setTimingConstraintValue

Description: Sets the given value for the given timingConstraint.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: timingConstraintID of type **String** - Handle to a timingConstraint element
- Input: value of type **String** - timingConstraint value

F.7.35 Constraint Set (BASE)

F.7.35.1 getConstraintSetTimingConstraints

Description: Returns all the timingConstraint values defined on the given constraintSet element.

- Returns: timingConstraintValues of type **Float** - List of timingConstraint values
- Input: constraintSetID of type **String** - Handle to a constraintSet element

F.7.36 Constraint Set (EXTENDED)

F.7.36.1 addConstraintSetTimingConstraint

Description: Adds a timingConstraint on the given constraintSet element

- Returns: timingConstraintID of type **String** - Handle to a timingConstraint element
- Input: constraintSetID of type **String** - Handle to a constraintSet element
- Input: value of type **Float** - The timingConstraint value
- Input: clockName of type **String** - The clockName value

F.7.37 Design (BASE)

F.7.37.1 getActiveinterfaceExcludePortIDs

Description: Returns the handles to all the excludePorts defined on the given activeInterface element.

- Returns: excludePortID of type **String** - List of handles to the excludePort elements
- Input: activeInterfaceID of type **String** - Handle to an activeInterface element

F.7.37.2 getAdHocConnectionExternalPortReferenceIDs

Description: Returns the handles to all the externalPortReferences defined on the given adHocConnection element.

- Returns: externalPortReferenceIDs of type **String** - List of handles to externalPortReference elements
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element

F.7.37.3 getAdHocConnectionInternalPortReferenceIDs

Description: Returns the handles to all the internalPortReferences defined on the given adHocConnection element.

- Returns: internalPortReferenceIDs of type **String** - List of handles to internalPortReference elements
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element

F.7.37.4 getAdHocConnectionTiedValue

Description: Returns tiedValue for the given adHocConnection element.

- Returns: value of type **String** - AdHoc connection tied value
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element

F.7.37.5 getAdHocConnectionTiedValueExpression

Description: Returns the tiedValue expression defined on the given adHocConnection element.

- Returns: valueExpression of type **String** - The tiedValue expression
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element

F.7.37.6 getAdHocConnectionTiedValueID

Description: Returns the handle to the tiedValue defined on the given adHocConnection element.

- Returns: `tiedValueID` of type ***String*** - Handle to the tiedValue element
- Input: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

F.7.37.7 getInternalPortReferenceComponentInstanceRefByName

Description: Returns the componentInstanceRef defined on the given internalPortReference element.

- Returns: `componentInstanceRef` of type ***String*** - The referenced componentInstance name
- Input: `internalPortReferenceID` of type ***String*** - Handle to an internalPortReference element

F.7.37.8 getComponentInstanceComponentRefByID

Description: Returns the handle to the componentreferenced from the given componentInstance element.

- Returns: `componentID` of type ***String*** - Handle to the referenced component object
- Input: `componentInstanceID` of type ***String*** - Handle to a componentInstantiation element

F.7.37.9 getComponentInstanceComponentRefByVLNV

Description: Returns the VLNV of the component referenced from the given componentInstance element.

- Returns: `VLNV` of type ***String*** - The VLNV of the referenced component object
- Input: `componentInstanceID` of type ***String*** - Handle to a componentInstantiation element

F.7.37.10 getComponentInstanceName

Description: Returns the name of the given componentInstance element.

- Returns: `name` of type ***String*** - The componentInstance name
- Input: `componentInstanceID` of type ***String*** - Handle to a componentInstance element

F.7.37.11 getComponentInstancePowerDomainLinkIDs

Description: Returns the handles to all the powerDomainLinks defined on the given componentInstance element.

- Returns: `powerDomainLinkIDs` of type ***String*** - List of handles to the powerDomainLink elements
- Input: `componentInstanceID` of type ***String*** - Handle to a componentInstance

F.7.37.12 getDesignAdHocConnectionIDs

Description: Returns the handles to all the adHocConnections defined on the given design object.

- Returns: `adHocConnectionIDs` of type ***String*** - List of handles to adHocConnection elements
- Input: `designID` of type ***String*** - Handle to a design object

F.7.37.13 getDesignChoiceIDs

Description: Returns the handles to all the choices defined on the given design object element.

- Returns: `choiceIDs` of type ***String*** - List of handles to choice elements
- Input: `designID` of type ***String*** - Handle to a design object element

F.7.37.14 getDesignComponentInstanceIDs

Description: Returns the handles to all the componentInstances defined on the given design object.

- Returns: `componentInstanceIDs` of type ***String*** - List of handles to componentInstance elements
- Input: `designID` of type ***String*** - Handle to a design object

F.7.37.15 getDesignID

Description: Returns the handle to the current or top design object associated with the currently invoked generator.

- Returns: `designID` of type ***String*** - Handle to current or top design object
- Input: `top` of type ***Boolean*** - Indicates if the call must return top (true) or current (false) designID

F.7.37.16 getDesignInterconnectionIDs

Description: Returns the handles to all the interconnections defined on the given design object.

- Returns: `interconnectionIDs` of type ***String*** - List of handles to interconnection elements
- Input: `designID` of type ***String*** - Handle to a design object

F.7.37.17 getDesignMonitorInterconnectionIDs

Description: Returns the handles to all the monitorInterconnections defined on the given design object.

- Returns: `monitorInterconnectionIDs` of type ***String*** - List of handles to monitorInterconnection elements
- Input: `designID` of type ***String*** - Handle to a design object

F.7.37.18 getExternalPortReferencePartSelectID

Description: Returns the handle to the partSelect element defined on the given externalPortReference element.

- Returns: `partSelectID` of type ***String*** - Handle to the partSelect element
- Input: `externalPortReferenceID` of type ***String*** - Handle to an externalPortReference element

F.7.37.19 getExternalPortReferencePortRefByName

Description: Returns the portRef expression from an externalPortReference element.

- Returns: `portRef` of type ***String*** - The portRef expression
- Input: `externalPortReferenceID` of type ***String*** - The externalPortReferenceID

F.7.37.20 getExternalPortReferenceSubPortReferenceIDs

Description: Returns the handles to all the subPortReferences defined on the given externalPortReference element.

- Returns: `subPortReferenceIDs` of type ***String*** - List of handles to the subPortReference elements
- Input: `externalPortReferenceID` of type ***String*** - Handle to an externalPortReference element

F.7.37.21 getInterconnectionActiveInterfaceIDs

Description: Returns the handles to all the activeInterfaces defined on the given interconnection element.

- Returns: `activeInterfaceIDs` of type ***String*** - List of handles to activeInterface element
- Input: `interconnectionID` of type ***String*** - Handle to an interconnection element

F.7.37.22 getInterconnectionHierInterfaceIDs

Description: Returns the handles to all the hierInterfaces defined on the given activeInterface element.

- Returns: `hierInterfaceIDs` of type ***String*** - List of handles to hierInterface elements
- Input: `interconnectionID` of type ***String*** - Handle to an interconnection element

F.7.37.23 getInternalPortReferencePartSelectID

Description: Returns the handle to the partSelect element defined on the given internalPortReference element.

- Returns: `partSelectID` of type ***String*** - Handle to the partSelect element
- Input: `internalPortReferenceID` of type ***String*** - Handle to an internalPortReference element

F.7.37.24 getInternalPortReferencePortRefByName

Description: Returns the portRef defined on the given internalPortReference element.

- Returns: `portRef` of type ***String*** - The referenced port name
- Input: `internalPortReferenceID` of type ***String*** - Handle to an internalPortReference element

F.7.37.25 getInternalPortReferenceSubPortReferenceIDs

Description: Returns the handles to all the subPortReferences defined on the given internalPortReference element.

- Returns: `subPortReferenceIDs` of type ***String*** - List of handles to the subPortReference elements
- Input: `internalPortReferenceID` of type ***String*** - Handle to an internalPortReference element

F.7.37.26 getMonitorInterconnectionMonitorInterfaceIDs

Description: Returns the handles to all the monitorInterfaces defined on the given monitorInterconnection element.

- Returns: `monitorInterfaceIDs` of type ***String*** - List of handles to the monitorInterface elements
- Input: `monitorInterconnectionID` of type ***String*** - Handle to a monitorInterconnection element

F.7.37.27 getMonitorInterconnectionMonitoredActiveInterfaceID

Description: Returns monitorActiveInterfaceID for the given monitorInterconnection element.

- Returns: `monitoredInterfaceID` of type ***String*** - Handle to a monitoredInterface

- Input: monitorInterconnectionID of type **String** - Handle to a monitorInterconnection element

F.7.37.28 getPowerDomainLinkExternalPowerDomainRef

Description: Returns the external power domain value from the given powerDomain link.

- Returns: externalRef of type **String** - The external power domain value
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element

F.7.37.29 getPowerDomainLinkExternalPowerDomainRefByID

Description: Returns the handle to the power domain referenced by the external reference from the given powerDomainLink element.

- Returns: powerDomainID of type **String** - Handle to the power domain
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element

F.7.37.30 getPowerDomainLinkExternalPowerDomainRefByName

Description: Returns the external power domain referenced from the given powerDomainLink element.

- Returns: externalRef of type **String** - The referenced external power domain
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element

F.7.37.31 getPowerDomainLinkExternalPowerDomainRefExpression

Description: Returns the external power domain expression from the given powerDomain link.

- Returns: externalRef of type **String** - The external power domain expression
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element

F.7.37.32 getPowerDomainLinkExternalPowerDomainRefID

Description: Returns the handle to the external power domain from the given powerDomain link.

- Returns: expressionID of type **String** - Handle to the external power domain element
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element

F.7.37.33 getPowerDomainLinkInternalPowerDomainRefs

Description: Returns all the internal power domains referenced from the given powerDomainLink element.

- Returns: internalRef of type **String** - List of the referenced internal power domains
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element

F.7.38 Design (EXTENDED)

F.7.38.1 addActiveInterfaceExcludePort

Description: add an excludePort on an active interface element

- Returns: excludePortID of type **String** - Handle to a new excludePort element
- Input: activeInterfaceID of type **String** - Handle to an active interface element
- Input: excludePort of type **String** - The excludePort name

F.7.38.2 addAdHocConnectionExternalPortReference

Description: Adds externalPortReference with the portRef to the given adHocConnection element.

- Returns: adHocExternalPortReferenceID of type **String** - Handle to an externalPortReference element
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element
- Input: portRef of type **String** - Port name

F.7.38.3 addAdHocConnectionInternalPortReference

Description: Adds internalPortReference with the given componentInstanceRef and portRef to the given adHocConnection element.

- Returns: adHocInternalPortReferenceID of type **String** - Handle to an internalPortReference element
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element
- Input: componentInstanceRef of type **String** - Component instance name
- Input: portRef of type **String** - Port name

F.7.38.4 addDesignAdHocConnection

Description: Adds adHocConnection with the given name, componentInstanceRef, and portRef to the given design element.

- Returns: adHocConnectionID of type **String** - Handle to an adHocConnection element
- Input: designID of type **String** - Handle to a design element
- Input: name of type **String** - AdHocConnection name
- Input: componentInstanceRef of type **String** - Component instance name
- Input: portRef of type **String** - Port name

F.7.38.5 addDesignChoice

Description: Adds a choice with the given name and enumerations to the given design element.

- Returns: choiceID of type **String** - Handle to a new choice
- Input: designID of type **String** - Handle to a design element
- Input: name of type **String** - Choice name
- Input: enumerations of type **String[]** - List of enumeration values

F.7.38.6 addDesignComponentInstance

Description: Adds componentInstance with the given VLVN and instance name to the given design element.

- Returns: componentInstanceID of type **String** - Handle to a component instance element
- Input: designID of type **String** - Handle to a design element
- Input: componentVLVN of type **String[]** - Component VLVN
- Input: componentInstanceName of type **String** - Component instance name

F.7.38.7 addDesignExternalAdHocConnection

Description: Adds adHocConnection with the given name and portRef to the given design element.

- Returns: adHocConnectionID of type **String** - Handle to an adHocConnection element

- Input: designID of type **String** - Handle to a design element
- Input: name of type **String** - AdHocConnection name
- Input: portRef of type **String** - Port name

F.7.38.8 addDesignInterconnection

Description: Adds interconnection with the given name between two component instance interfaces given by ComponentInstanceRef1, busRef1, componentInstanceRef2, and busInterfaceRef2 to the given design element.

- Returns: interconnectionID of type **String** - Handle to an interconnection element
- Input: designID of type **String** - Handle to a design element
- Input: name of type **String** - Interconnection name
- Input: componentInstanceRef1 of type **String** - Component instance name
- Input: busInterfaceRef1 of type **String** - Bus interface name
- Input: componentInstanceRef2 of type **String** - Component instance name
- Input: busInterfaceRef2 of type **String** - Bus interface name

F.7.38.9 addDesignMonitorInterconnection

Description: Adds monitorInterconnection with the given name, componentInstanceRef, and busRef to the given design object.

- Returns: monitorInterconnectionID of type **String** - Handle to a monitorInterconnection element.
- Input: designID of type **String** - Handle to a design object
- Input: name of type **String** - MonitorInterconnection name
- Input: componentInstanceRef1 of type **String** - Component instance name
- Input: activeInterfaceBusInterfaceRef of type **String** - Bus interface name
- Input: componentInstanceRef2 of type **String** - Component instance name
- Input: interfaceBusInterfaceRef of type **String** - Bus interface name

F.7.38.10 addInterconnectionActiveInterface

Description: Adds activeInterface with the given componentInstanceRef and busRef to the given interconnection element.

- Returns: activeInterfaceID of type **String** - Handle to an activeInterface element
- Input: interconnectionID of type **String** - Handle to an interconnection element
- Input: componentInstanceRef of type **String** - Component instance name
- Input: busInterfaceRef of type **String** - Bus interface name

F.7.38.11 addInterconnectionHierInterface

Description: Adds hierInterface with the given busRef to the given interconnection element.

- Returns: hierInterfaceID of type **String** - Handle to an hierInterface element
- Input: interconnectionID of type **String** - Handle to an interconnection element
- Input: busInterfaceRef of type **String** - Bus interface name

F.7.38.12 addMonitorInterconnectionMonitorInterface

Description: Adds monitorInterface with the given componentInstanceRef and busRef to the given monitorInterconnection element.

- Returns: monitorInterfaceID of type **String** - Handle to a monitorInterface element
- Input: monitorInterconnectionID of type **String** - Handle to a monitorInterconnection element
- Input: componentInstanceRef of type **String** - Component instance name
- Input: busInterfaceRef of type **String** - Bus interface name

F.7.38.13 removeActiveInterfaceExcludePort

Description: Removes an excludePort on an active interface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: excludePortID of type **String** - Handle to an excludePort element

F.7.38.14 removeAdHocConnectionExternalPortReference

Description: Removes the given externalPortReference element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: externalPortReferenceID of type **String** - Handle to an externalPortReference element

F.7.38.15 removeAdHocConnectionInternalPortReference

Description: Removes the given internalPortReference element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: internalPortReferenceID of type **String** - Handle to an internalPortReference element

F.7.38.16 removeAdHocConnectionTiedValue

Description: Removes tiedValue from the given adHocConnection element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element

F.7.38.17 removeDesignAdHocConnection

Description: Removes the given adHocConnection element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element

F.7.38.18 removeDesignChoice

Description: Removes the given choice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: choiceID of type **String** - Handle to a choice element

F.7.38.19 removeDesignComponentInstance

Description: Removes the given componentInstance element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstanceID of type **String** - Handle to a componentInstance element

F.7.38.20 removeDesignInterconnection

Description: Removes the given interconnection element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: interconnectionID of type **String** - Handle to an interconnection element

F.7.38.21 removeDesignMonitorConnection

Description: Removes the given monitorInterconnection element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: monitorInterconnectionID of type **String** - Handle to a monitorInterconnection element

F.7.38.22 removeExternalPortReferencePartSelect

Description: Removes a partSelect on the given externalPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: externalPortReferenceID of type **String** - Handle to a portMap externalPortReference element

F.7.38.23 removeExternalPortReferenceSubPortReference

Description: Removes the given subPort reference from the external PortReference of a structured port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subPortRefID of type **String** - Handle to an externalPortReference element

F.7.38.24 removeInterconnectionActiveInterface

Description: Removes the given activeInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: activeInterfaceID of type **String** - Handle to an activeInterface element

F.7.38.25 removeInterconnectionHierInterface

Description: Removes the given hierInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: hierInterfaceID of type **String** - Handle to an hierInterface element

F.7.38.26 removeInternalPortReferencePartSelect

Description: Removes a partSelect on the given internalPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: internalPortReferenceID of type **String** - Handle to a portMap internalPortReference element

F.7.38.27 removeInternalPortReferenceSubPortReference

Description: Removes the given subPort reference from the internalPortReference of a structured port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subPortRefID of type **String** - Handle to an subPortRef element

F.7.38.28 removeMonitorInterconnectionMonitorInterface

Description: Removes the given monitorInterface interface.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: monitorInterfaceID of type **String** - Handle to a monitorInterface element

F.7.38.29 setAdHocConnectionTiedValue

Description: Sets tiedValue with the given value for the given adHocConnection element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: adHocConnectionID of type **String** - Handle to an adHocConnection element
- Input: tiedValue of type **String** - AdHocConnection tied value

F.7.38.30 setComponentInstanceComponentRef

Description: Sets the componentRef with the given value for the given componentInstance element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstanceID of type **String** - Handle to an componentInstance element
- Input: componentVNV of type **String[]** - component reference

F.7.38.31 setExternalPortReferencePartSelect

Description: Sets a partSelect on the given externalPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: externalPortReferenceID of type **String** - Handle to a portMap externalPortReference element
- Input: range of type **String[]** - Create the range on the partSelect with “left” for range[0] and “right” for range[1]. Set to null if you only want indices.
- Input: indices of type **String[]** - Handle to values of type String. Set all the index on the partSelect

F.7.38.32 setInternalPortReferencePartSelect

Description: Sets a partSelect on the given internalPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: internalPortReferenceID of type **String** - Handle to a portMap internalPortReference element
- Input: range of type **String[]** - Create the range on the partSelect with “left” for range[0] and “right” for range[1]. Set to null if you only want indices.
- Input: indices of type **String[]** - Handle to values of type String. Set all the index on the partSelect

F.7.38.33 setInterconnectionActiveInterface

Description: Sets the active interface of an interconnection.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: interconnectionID of type **String** - Handle to interconnection
- Input: componentInstanceRef of type **String** - Component interface name
- Input: busRef of type **String** - Bus interface name

F.7.38.34 setInterconnectionHierInterface

Description: Sets the hierarchical interface of an interconnection.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: interconnectionID of type **String** - Handle to interconnection
- Input: busRef of type **String** - Bus interface name

F.7.38.35 setMonitorInterconnectionMonitoredActiveInterface

Description: Sets the monitoredActiveInterface on a monitorInterconnection element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: monitorInterconnectionID of type **String** - Handle to a monitorInterconnection element
- Input: componentInstanceRef of type **String** - The componentInstanceRef to be set
- Input: busRef of type **String** - The busRef attribute to be set

F.7.38.36 setMonitoredActiveInterfacePath

Description: Sets path with the given value for the given monitoredActiveInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: monitoredActiveInterfaceID of type **String** - Handle to a monitoredActiveInterface element
- Input: path of type **String** - Hierarchical path separated by a slash

F.7.38.37 setPowerDomainLinkExternalPowerDomainRef

Description: Sets the external power domain reference to the expression by the given powerDomain link.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element
- Input: expression of type **String** - The new value or expression

F.7.39 Design configuration (BASE)

F.7.39.1 getAbstractorInstanceAbstractorRefByID

Description: Returns the handle to the abstractor instance referenced from the given abstractorInstance element.

- Returns: abstractorID of type **String** - Handle to the referenced abstractor object
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element

F.7.39.2 getAbstractorInstanceAbstractorRefByVLNV

Description: Returns the VLNV of the abstractor referenced from the given abstractorInstance element.

- Returns: VLNV of type **String** - The VLNV of the referenced abstractor object
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element

F.7.39.3 getAbstractorInstanceInstanceName

Description: Returns the instanceName defined on the given abstractorInstance element.

- Returns: instanceName of type **String** - The instance name
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element

F.7.39.4 getAbstractorInstanceViewName

Description: Returns viewName for the given abstractorInstance element.

- Returns: viewName of type **String** - Abstractor view name
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element

F.7.39.5 getAbstractorInstancesAbstractorInstanceIDs

Description: Returns the handles to all the abstractorInstances defined on the given abstractorInstances element.

- Returns: abstractorInstanceIDs of type **String** - List of handles to abstractorInstance elements
- Input: abstractorInstancesID of type **String** - Handle to an abstractorInstances element

F.7.39.6 getAbstractorInstancesInterfaceRefIDs

Description: Returns the handles to all the abstractorInstancesInterfaces defined on the given abstractorInstances element.

- Returns: abstractorInstancesInterfaceIDs of type **String** - List of handles to the abstractorInstancesInterface elements
- Input: abstractorInstancesID of type **String** - Handle to an abstractorInstances element

F.7.39.7 getDesignConfigurationChoiceIDs

Description: Returns the handles to all the choices defined on the given designConfiguration element.

- Returns: choiceIDs of type **String** - List of handles to choice elements
- Input: designConfigurationID of type **String** - Handle to a designConfiguration element

F.7.39.8 getDesignConfigurationDesignRefByID

Description: Returns the handle to the design instance referenced in the given designConfiguration element.

- Returns: designID of type **String** - Handle to the referenced design object
- Input: designConfigurationID of type **String** - Handle to a designConfiguration element

F.7.39.9 getDesignConfigurationDesignRefByVLNV

Description: Returns the VLNV of the design referenced from the given designConfiguration element.

- Returns: VLNV of type **String** - The VLNV of the referenced design object

- Input: designConfigurationID of type **String** - Handle to a designConfiguration element

F.7.39.10 getDesignConfigurationGeneratorChainConfigurationIDs

Description: Returns the handles to all the generatorChainConfigurations defined on the given designConfiguration object.

- Returns: generatorChainConfigurationIDs of type **String** - List of handles to generatorChainConfiguration elements
- Input: designConfigurationID of type **String** - Handle to a designConfiguration object

F.7.39.11 getDesignConfigurationInterconnectionConfigurationIDs

Description: Returns the handles to all the interconnectionConfigurations defined on the given designConfiguration object.

- Returns: interconnectionConfigurationIDs of type **String** - List of handles to interconnectionConfiguration elements
- Input: designConfigurationOrDesignConfigurationInstanceID of type **String** - Handle to a designConfiguration or designConfigurationInstance element

F.7.39.12 getDesignConfigurationViewConfigurationIDs

Description: Returns the handles to all the viewConfigurations defined on the given designConfiguration object.

- Returns: viewConfigurationIDs of type **String** - List of handles to viewConfiguration elements
- Input: designConfigurationOrDesignConfigurationInstanceID of type **String** - Handle to a designConfiguration or designConfigurationInstance element

F.7.39.13 getGeneratorChainConfigurationRefByID

Description: Returns the handle to the generatorChain instance referenced from the given generatorChainConfiguration element.

- Returns: generatorChainID of type **String** - Handle to the referenced generatorChain object
- Input: generatorChainConfigurationID of type **String** - Handle to a generatorChainConfiguration element

F.7.39.14 getGeneratorChainConfigurationRefByVLNV

Description: Returns the VLNV of the generatorChainConfiguration referenced from the given generatorChainConfiguration element.

- Returns: generatorChainConfigurationVLNV of type **String** - The VLNV of generatorChainConfiguration
- Input: generatorChainConfigurationID of type **String** - Handle to a generatorChainConfiguration element

F.7.39.15 getInterconnectionConfigurationAbstractorsInstancesIDs

Description: Returns the handles to all the abstractorInstances defined on the given interconnectionConfiguration element.

- Returns: abstractorInstancesIDs of type **String** - List of handles to the abstractorInstances elements

- Input: interconnectionConfigurationID of type **String** - Handle to an interconnectConfiguration element

F.7.39.16 getInterconnectionConfigurationInterconnectionRefByID

Description: Returns the handle to the interconnection referenced from the given interconnectionConfiguration element.

- Returns: interconnectionID of type **String** - Handle to the referenced interconnection
- Input: interconnectionConfigurationID of type **String** - Handle to an interconnectConfiguration element

F.7.39.17 getInterconnectionConfigurationInterconnectionRefByName

Description: Returns the interconnectionRef defined on the given interconnectionConfiguration element.

- Returns: interconnectionRef of type **String** - The referenced interconnection
- Input: interconnectionConfigurationID of type **String** - Handle to an interconnectionConfiguration element

F.7.39.18 getViewConfigurationConfigurableElementValueIDs

Description: Returns the handles to all the configurableElementValues defined on the given viewConfiguration element.

- Returns: configurableElementValueIDs of type **String** - List of handles to configurableElementValue elements
- Input: viewConfigurationID of type **String** - Handle to a viewConfiguration element

F.7.39.19 getViewConfigurationInstanceName

Description: Returns instanceName for the given viewConfiguration element.

- Returns: instanceName of type **String** - Component instance name
- Input: viewConfigurationID of type **String** - Handle to a viewConfiguration element

F.7.39.20 getViewConfigurationViewID

Description: Returns the handle to the view defined on the given viewConfiguration element.

- Returns: viewID of type **String** - Handle to the view element
- Input: viewConfigurationID of type **String** - Handle to a viewConfiguration element

F.7.39.21 getViewConfigurationViewRefByName

Description: Returns the viewRef defined on the given viewConfiguration element.

- Returns: viewRef of type **String** - The referenced view
- Input: viewConfigurationID of type **String** - Handle to a view element

F.7.40 Design configuration (EXTENDED)

F.7.40.1 addAbstractorInstancesAbstractorInstance

Description: Adds abstractorInstance with the given VLVN, instance, and viewName to the given abstractorInstances element.

- Returns: abstractorInstanceID of type **String** - Handle to an abstractorInstantiations element
- Input: abstractorInstancesID of type **String** - Handle to an abstractorInstances element
- Input: abstractorVNV of type **String[]** - Abstractor VNV
- Input: instanceName of type **String** - Abstractor instance name
- Input: viewName of type **String** - Abstractor instance view name

F.7.40.2 addAbstractorInstancesInterfaceRef

Description: Adds abstractorInstancesInterface with the given componentRef and busRef to the given abstractorInstances element.

- Returns: interfaceRef of type **String** - Handle to the new interfaceRef element
- Input: abstractorInstancesID of type **String** - Handle to an abstractorInstances element
- Input: componentInstanceName of type **String** - The componentInstance name
- Input: busRef of type **String** - BusInterface name

F.7.40.3 addDesignConfChoice

Description: Adds a choice with the given name and enumerations to the given design configuration element.

- Returns: choiceID of type **String** - Handle to a new choice
- Input: designConfID of type **String** - Handle to a design configuration element
- Input: name of type **String** - Choice name
- Input: enumerations of type **String[]** - List of enumeration values

F.7.40.4 addDesignConfigurationGeneratorChainConfiguration

Description: Adds generatorChainConfiguration with the given VNV to the given designConfiguration element.

- Returns: generatorChainConfigurationID of type **String** - Handle to a generatorChainConfiguration element
- Input: designConfigurationID of type **String** - Handle to a designConfiguration element
- Input: generatorVNV of type **String[]** - Generator VNV

F.7.40.5 addDesignConfigurationInterconnectionConfiguration

Description: Adds interconnectionConfiguration with the given interconnectionRef, VNV, instance, and viewName to the given designConfiguration element.

- Returns: interconnectionConfigurationID of type **String** - Handle to an interconnectionConfiguration
- Input: designConfigurationID of type **String** - Handle to a designConfiguration element
- Input: interconnectionRef of type **String** - Interconnection name
- Input: abstractorVNV of type **String[]** - Abstractor VNV
- Input: instanceName of type **String** - Abstractor instance name
- Input: viewName of type **String** - Abstractor instance view name

F.7.40.6 addDesignConfigurationViewConfiguration

Description: Adds viewConfiguration with the given instanceName and viewName to the given designConfiguration element.

- Returns: viewConfigurationID of type **String** - Handle to a viewConfiguration element
- Input: designConfigurationID of type **String** - Handle to a designConfiguration element
- Input: componentInstanceName of type **String** - ComponentInstance name
- Input: viewName of type **String** - Component view name

F.7.40.7 addInterconnectionConfigurationAbstractorInstances

Description: Adds abstractorInstances with the given VLNV, instance, and viewName to the given interconnectionConfiguration element.

- Returns: abstractorInstancesID of type **String** - Handle to an abstractorInstances element
- Input: interconnectionConfigurationID of type **String** - Handle to an interconnectConfiguration element
- Input: abstractorVLNV of type **String[]** - Abstractor VLNV
- Input: instanceName of type **String** - Abstractor instance name
- Input: viewName of type **String** - Abstractor instance view name

F.7.40.8 removeAbstractorInstancesAbstractorInstance

Description: Removes the given abstractorInstance element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractorInstancesID of type **String** - Handle to an abstractorInstances element

F.7.40.9 removeAbstractorInstancesInterfaceRef

Description: Removes the given abstractorInstancesInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: interfaceRefID of type **String** - Handle to an interfaceRef element

F.7.40.10 removeDesignConfChoice

Description: Removes the given choice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: choiceID of type **String** - Handle to a choice element

F.7.40.11 removeDesignConfigurationDesignRef

Description: Removes designRef from the given designConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: designConfigurationID of type **String** - Handle to a designConfiguration element

F.7.40.12 removeDesignConfigurationGeneratorChainConfiguration

Description: Removes the given generatorChainConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: generatorChainConfigurationID of type **String** - Handle to a generatorChainConfigurationElement

F.7.40.13 removeDesignConfigurationInterconnectionConfiguration

Description: Removes the given interconnectionConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: interconnectionConfigurationID of type **String** - Handle to an interconnectConfiguration element

F.7.40.14 removeDesignConfigurationViewConfiguration

Description: Removes the given viewConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewConfigurationID of type **String** - Handle to a viewConfiguration element

F.7.40.15 removeInterconnectionConfigurationAbstractorInstances

Description: Removes the given abstractorInstances element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractorInstancesID of type **String** - Handle to an abstractorInstances element

F.7.40.16 removeViewConfigurationConfigurableElementValue

Description: Removes the given configurable element value from its containing viewConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: configurableElementValueID of type **String** - Handle to the configurableElementValue

F.7.40.17 setAbstractorInstanceAbstractorRef

Description: Sets the abstractorRef on an abstractorInstance element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element
- Input: vlnv of type **String[]** - The vlnv to be set on the abstractorRef

F.7.40.18 setAbstractorInstanceInstanceName

Description: Sets the instanceName on an abstractorInstance element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element
- Input: instanceName of type **String** - The instanceName expression

F.7.40.19 setAbstractorInstanceViewName

Description: Sets the viewName on an abstractorInstance element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element
- Input: viewName of type **String** - The viewname expression

F.7.40.20 setDesignConfigurationDesignRef

Description: Sets designRef with the given VLVN for the given designConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: designConfigurationID of type **String** - Handle to a designConfiguration element
- Input: designVLNV of type **String[]** - Design VLVN

F.7.40.21 setInterconnectionConfigurationInterconnectionRef

Description: get the interconnectionRef on an interconnectionConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: interconnectionConfigurationID of type **String** - Handle to an interconnectionConfiguration element
- Input: interconnectionRef of type **String** - The interconnectionRef expression

F.7.40.22 setViewConfigurationInstanceName

Description: Sets the instanceName on a viewConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewConfigurationID of type **String** - Handle to a viewConfiguration element
- Input: instanceName of type **String** - The instanceName expression

F.7.40.23 setViewConfigurationView

Description: Sets the view on a viewConfiguration element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewConfigurationID of type **String** - Handle to a viewConfiguration element
- Input: viewRef of type **String** - the viewRef expression

F.7.41 Driver (BASE)

F.7.41.1 getClockDriverClockPeriod

Description: Returns the clockPeriod defined on the given clockDriver element.

- Returns: clockPeriod of type **Double** - The clock period value
- Input: clockDriverID of type **String** - Handle to a clockDriver element

F.7.41.2 getClockDriverClockPeriodExpression

Description: Returns the clockPeriod expression defined on the given clockDriver element.

- Returns: clockPeriodExpression of type **String** - The clockPeriod expression
- Input: clockDriverID of type **String** - Handle to a clockDriver element

F.7.41.3 getClockDriverClockPeriodID

Description: Returns the handle to the clockPeriod defined on the given clockDriver element.

- Returns: clockPeriodID of type **String** - Handle to the clockPeriod element
- Input: clockDriverID of type **String** - Handle to a clockDriver element

F.7.41.4 getClockDriverClockPulseDuration

Description: Returns the clockPulseDuration defined on the given clockDriver element.

- Returns: `clockPulseDuration` of type **Double** - The clock pulse duration value
- Input: `clockDriverID` of type **String** - Handle to a clockDriver element

F.7.41.5 getClockDriverClockPulseDurationExpression

Description: Returns the clockPulseDuration expression defined on the given clockDriver element.

- Returns: `clockPulseDurationExpression` of type **String** - The clockPulseDuration expression
- Input: `clockDriverID` of type **String** - Handle to a clockDriver element

F.7.41.6 getClockDriverClockPulseDurationID

Description: Returns the handle to the pulseDuration defined on the given clockDriver element.

- Returns: `pulseDurationID` of type **String** - Handle to the pulseDuration element
- Input: `clockDriverID` of type **String** - Handle to a clockDriver element

F.7.41.7 getClockDriverClockPulseOffset

Description: Returns the clockPulseOffset defined on the given clockDriver element.

- Returns: `clockPulseOffset` of type **Double** - The clock pulse offset value
- Input: `clockDriverID` of type **String** - Handle to a clockDriver element

F.7.41.8 getClockDriverClockPulseOffsetExpression

Description: Returns the clockPulseOffset expression defined on the given clockDriver element.

- Returns: `clockPulseOffsetExpression` of type **String** - The clockPulseOffset expression
- Input: `clockDriverID` of type **String** - Handle to a clockDriver element

F.7.41.9 getClockDriverClockPulseOffsetID

Description: Returns the handle to the pulseOffset defined on the given clockDriver element.

- Returns: `pulseOffsetID` of type **String** - Handle to the pulseOffset element
- Input: `clockDriverID` of type **String** - Handle to a clockDriver element

F.7.41.10 getClockDriverClockPulseValue

Description: Returns the clockPulseValue defined on the given clockDriver element.

- Returns: `clockPulseValue` of type **Boolean** - The clock pulse value
- Input: `clockDriverID` of type **String** - Handle to a clockDriver element

F.7.41.11 getClockDriverClockPulseValueExpression

Description: Returns the clockPulseValue expression defined on the given clockDriver element.

- Returns: `clockPulseValueExpression` of type **String** - The clockPulseValue expression
- Input: `clockDriverID` of type **String** - Handle to a clockDriver element

F.7.41.12 getClockDriverClockPulseValueID

Description: Returns the handle to the pulseValue defined on the given clockDriver element.

- Returns: `pulseValueID` of type ***String*** - Handle to the pulseValue element
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

F.7.41.13 getDriverClockDriverID

Description: Returns the handle to the clockDriver defined on the given driver element.

- Returns: `clockDriverID` of type ***String*** - Handle to the clockDriver
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.14 getDriverDefaultValue

Description: Returns the default value defined on the given driver element.

- Returns: `defaultValue` of type ***String*** - The driver default value
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.15 getDriverDefaultValueExpression

Description: Returns the defaultValue expression defined on the given driver element.

- Returns: `defaultValueExpression` of type ***String*** - The defaultValue expression
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.16 getDriverDefaultValueID

Description: Returns the handle to the defaultValue defined on the given driver element.

- Returns: `defaultValueID` of type ***String*** - Handle to the defaultValue element
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.17 getDriverLeftID

Description: Returns the handle to the left range defined the given driver element.

- Returns: `leftID` of type ***String*** - Handle to the left range element
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.18 getDriverRange

Description: Returns the range left and right (resolved) values defined on the given driver element.

- Returns: `rangeValues` of type ***Long*** - Array of two range values: left and right
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.19 getDriverRangeExpression

Description: Returns the range left and right expressions defined on the given driver element.

- Returns: `rangeExpressions` of type ***String*** - Array of two range expressions: left and right
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.20 getDriverRightID

Description: Returns the handle to the right range defined the given driver element.

- Returns: `rightID` of type ***String*** - Handle to the right range element
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.21 getDriverSingleShotDriverID

Description: Returns the handle to the singleShotDriver element defined on the given driver element.

- Returns: `singleShotDriverID` of type ***String*** - Handle to the singleShotDriver element
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.22 getDriverViewRefByID

Description: Returns the handle to the view defined on the driver element.

- Returns: `viewID` of type ***String*** - Handles to a view element
- Input: `driverID` of type ***String*** - Handle to a driver element
- Input: `viewRef` of type ***String*** - Handle to the viewRef element

F.7.41.23 getDriverViewRefIDs

Description: Returns the handles to all the viewRefs defined on the driver element.

- Returns: `viewRefIDs` of type ***String*** - List of handles to the viewRef elements
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.41.24 getOtherClockDriverClockPeriod

Description: Returns the `clockPeriod` value defined on the given otherClockDriver element.

- Returns: `clockPeriod` of type ***Double*** - The `clockPeriod` value
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.25 getOtherClockDriverClockPeriodExpression

Description: Returns the `clockPeriod` expression defined on the given otherClockDriver element.

- Returns: `clockPeriod` of type ***String*** - The `clockPeriod` expression
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.26 getOtherClockDriverClockPeriodID

Description: Returns the handle to the `clockPeriod` defined on the given otherClockDriver element.

- Returns: `clockPeriodID` of type ***String*** - Handle to the `clockPeriod` element
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.27 getOtherClockDriverClockPulseDuration

Description: Returns the `clockPulseDuration` value defined on the given otherClockDriver element.

- Returns: `clockPulseDuration` of type ***Double*** - The `clockPulseDuration` value
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.28 getOtherClockDriverClockPulseDurationExpression

Description: Returns the clockPulseDuration expression defined on the given otherClockDriver element.

- Returns: `clockPulseDurationExpression` of type ***String*** - The clockPulseDuration expression
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.29 getOtherClockDriverClockPulseDurationID

Description: Returns the handle to the clockPulseDuration defined on the given otherClockDriver element.

- Returns: `clockPulseDurationID` of type ***String*** - Handle to the clockPulseDuration element
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.30 getOtherClockDriverClockPulseOffset

Description: Returns the clockPulseOffset value defined on the given otherClockDriver element.

- Returns: `clockPulseOffset` of type ***Double*** - The clockPulseOffset value
- Input: `otherClockDriverID` of type ***String*** - Handle to a otherClockDriver element

F.7.41.31 getOtherClockDriverClockPulseOffsetExpression

Description: Returns the clockPulseOffset expression defined on the given otherClockDriver element.

- Returns: `clockPulseOffsetExpression` of type ***String*** - The clockPulseOffset expression
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.32 getOtherClockDriverClockPulseOffsetID

Description: Returns the handle to the clockPulseOffset defined on the given otherClockDriver element.

- Returns: `clockPulseOffsetID` of type ***String*** - Handle to the clockPulseOffset element
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.33 getOtherClockDriverClockPulseValue

Description: Returns the clockPulseValue value defined on the given otherClockDriver element.

- Returns: `clockPulseValue` of type ***Boolean*** - The clockPulseValue value
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.34 getOtherClockDriverClockPulseValueExpression

Description: Returns the clockPulseValue expression defined on the given otherClockDriver element.

- Returns: `clockPulseValueExpression` of type ***String*** - The clockPulseValue expression
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.35 getOtherClockDriverClockPulseValueID

Description: Returns the handle to the clockPulseValue defined on the given otherClockDriver element.

- Returns: `clockPulseValueID` of type ***String*** - Handle to the clockPulseValue element
- Input: `otherClockDriverID` of type ***String*** - Handle to an otherClockDriver element

F.7.41.36 getSingleShotDriverSingleShotDuration

Description: Returns the singleShotDuration defined on the given singleShotDriver element.

- Returns: `singleShotDuration` of type **Double** - The single shot duration
- Input: `singleShotDriverID` of type **String** - Handle to a singleShotDriver element

F.7.41.37 getSingleShotDriverSingleShotDurationExpression

Description: Returns the singleShotDuration expression defined on the given singleShotDriver element.

- Returns: `singleShotDurationExpression` of type **String** - The singleShotDuration expression
- Input: `singleShotDriverID` of type **String** - Handle to a singleShotDriver element

F.7.41.38 getSingleShotDriverSingleShotDurationID

Description: Returns the handle to the singleShotDuration element defined on the given singleShotDriver element.

- Returns: `singleShotDurationID` of type **String** - Handle to the singleShotDuration element
- Input: `singleShotDriverID` of type **String** - Handle to a singleShotDriver element

F.7.41.39 getSingleShotDriverSingleShotOffset

Description: Returns the singleShotOffset defined on the given singleShotDriver element.

- Returns: `singleShotOffset` of type **Double** - The single shot offset value
- Input: `singleShotDriverID` of type **String** - Handle to a singleShotDriver element

F.7.41.40 getSingleShotDriverSingleShotOffsetExpression

Description: Returns the singleShotOffset expression defined on the given singleShotDriver element.

- Returns: `singleShotOffsetExpression` of type **String** - The singleShotOffset expression
- Input: `singleShotDriverID` of type **String** - Handle to a singleShotDriver element

F.7.41.41 getSingleShotDriverSingleShotOffsetID

Description: Returns the handle to singleShotOffset element defined on the given singleShotDriver element.

- Returns: `singleShotOffsetID` of type **String** - Handle to the singleShotOffset element
- Input: `singleShotDriverID` of type **String** - Handle to a singleShotDriver element

F.7.41.42 getSingleShotDriverSingleShotValue

Description: Returns the singleShotValue defined on the given singleShotDriver element.

- Returns: `singleShotValue` of type **Long** - The single shot value
- Input: `singleShotDriverID` of type **String** - Handle to a singleShotDriver element

F.7.41.43 getSingleShotDriverSingleShotValueExpression

Description: Returns the singleShotValue expression defined on the given singleShotDriver element.

- Returns: `singleShotValueExpression` of type **String** - The singleShotValue expression
- Input: `singleShotDriverID` of type **String** - Handle to a singleShotDriver element

F.7.41.44 getSingleShotDriverSingleShotValueID

Description: Returns the handle to the singleShotValue element defined on the given singleShotDriver element.

- Returns: `singleShotValueID` of type ***String*** - Handle to the singleShotValue element
- Input: `singleShotDriverID` of type ***String*** - Handle to a singleShotDriver element

F.7.42 Driver (EXTENDED)

F.7.42.1 addDriverSingleShotDriver

Description: Adds a single shot driver with the given details to the given driver element.

- Returns: `singleShotDriverID` of type ***String*** - Handle to a new singleShotDriver
- Input: `driverID` of type ***String*** - Handle to a driver element
- Input: `offset` of type ***String*** - Single shot offset expression
- Input: `value` of type ***String*** - Single shot value expression
- Input: `duration` of type ***String*** - Single shot duration expression

F.7.42.2 addDriverViewRef

Description: Adds a reference to a view name in the file for which this type applies.

- Returns: `viewRefID` of type ***String*** - Handle to the added viewRef
- Input: `driverID` of type ***String*** - Handle to a driver element
- Input: `viewRef` of type ***String*** - Referenced view name

F.7.42.3 removeDriverClockDriver

Description: Removes the clock driver from the given driver element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.42.4 removeDriverRange

Description: Removes the range from the given driver.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.42.5 removeDriverSingleShotDriver

Description: Removes the single shot driver from the given driver element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.42.6 removeDriverViewRef

Description: Removes the given viewRef from its containing Driver element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `viewRefID` of type ***String*** - Handle to a viewRef element

F.7.42.7 setClockDriverClockPeriod

Description: Sets the clockPeriod on the given clockDriver element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: clockDriverID of type **String** - Handle to a clockDriver element
- Input: clockPeriod of type **String** - The clockPeriod expression

F.7.42.8 setClockDriverClockPulseDuration

Description: Sets the clockPulseDuration on the given clockDriver element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: clockDriverID of type **String** - Handle to a clockDriver element
- Input: clockPulseDuration of type **String** - The clockPulseDuration expression

F.7.42.9 setClockDriverClockPulseOffset

Description: Sets the pulseOffset on the given clockDriver element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: clockDriverID of type **String** - Handle to a clockDriver element
- Input: pulseOffset of type **String** - The pulseOffset expression

F.7.42.10 setClockDriverClockPulseValue

Description: Sets the clockPulseValue on the given clockDriver element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: clockDriverID of type **String** - Handle to a clockDriver element
- Input: clockPulseValue of type **String** - The clockPulseValue expression

F.7.42.11 setDriverClockDriver

Description: Sets the clockDriver on the given driver element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: driverID of type **String** - Handle to a driver element
- Input: clockPeriod of type **String** - The clockPeriod expression
- Input: clockPulseOffset of type **String** - The clockPulseOffset expression
- Input: clockPulseValue of type **String** - The clockPulseValue expression
- Input: clockPulseDuration of type **String** - The clockPulseDuration expression

F.7.42.12 setDriverSingleShotDriver

Description: Sets the singleShotDriver on the given driver element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: driverID of type **String** - Handle to a driver element
- Input: singleShotOffset of type **String** - The singleShotOffset expression
- Input: singleShotValue of type **String** - The singleShotValue expression
- Input: singleShotDuration of type **String** - The singleShotDuration expression

F.7.42.13 setDriverDefaultValue

Description: Sets a Default value for a wire port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: driverID of type **String** - Handle to a driver element
- Input: value of type **String** - Default driver value

F.7.42.14 setDriverRange

Description: Sets the range of the given driver.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: driverID of type **String** - Handle to a driver element
- Input: range of type **String[]** - Range with left at index 0 and right at index 1

F.7.42.15 setOtherClockDriverClockPeriod

Description: Sets a clockPeriod element on an otherClockDriver element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: otherClockDriverID of type **String** - Handle to an otherClockDriver element
- Input: clockPeriod of type **String** - Clock period value

F.7.42.16 setOtherClockDriverClockPulseDuration

Description: Sets a clockPulseDuration element on an otherClockDriver element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: otherClockDriverID of type **String** - Handle to an otherClockDriver element
- Input: clockPulseDuration of type **String** - Clock pulse offset value

F.7.42.17 setOtherClockDriverClockPulseOffset

Description: Sets a clockPulseOffset element on an otherClockDriver element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: otherClockDriverID of type **String** - Handle to an otherClockDriver element
- Input: clockPulseOffset of type **String** - Clock pulse offset value

F.7.42.18 setOtherClockDriverClockPulseValue

Description: Sets a clockPulseValue element on an otherClockDriver element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: otherClockDriverID of type **String** - Handle to an otherClockDriver element
- Input: clockPulseValue of type **String** - Clock pulse offset value

F.7.42.19 setSingleShotDriverSingleShotDuration

Description: Sets the singleShotDuration on the given singleShotDriver element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: singleShotDriverID of type **String** - Handle to a singleShotDuration element
- Input: singleShotDuration of type **String** - The singleShotDuration expression

F.7.42.20 setSingleShotDriverSingleShotOffset

Description: Sets the singleShotOffset on the given singleShotDriver element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: singleShotDriverID of type **String** - Handle to a singleShotDriver element
- Input: singleShotOffset of type **String** - The singleShotOffset expression

F.7.42.21 setSingleShotDriverSingleShotValue

Description: Sets the singleShotValue on the given singleShotDriver element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: singleShotDriverID of type **String** - Handle to a singleShotDriver element
- Input: singleShotValue of type **String** - The singleShotValue expression

F.7.43 Element attribute (BASE)

F.7.43.1 getAddressBlockRefAttribute

Description: Returns the attribute “addressBlockRef” defined on the given addressBlockRef element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.2 getAddressBlockRefAttributeByID

Description: Returns the handle to the addressBlock referenced from the given element.

- Returns: addressBlockID of type **String** - Handle to the referenced addressBlock
- Input: addressBlockRefID of type **String** - Handle to an element

F.7.43.3 getAddressSpaceRefAttribute

Description: Returns the attribute “addressSpaceRef” defined on the given element

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.4 getAddressSpaceRefAttributeByID

Description: Returns the handle to the addressSpace referenced from the attribute of the given element.

- Returns: addressSpaceID of type **String** - Handle to the referenced addressSpace
- Input: elementID of type **String** - Handle to an element

F.7.43.5 getAllBitsBooleanAttribute

Description: Returns the attribute “allBits” defined on the given wirePort width element.

- Returns: attributeValue of type **Boolean** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.6 getAllLogicalDirectionsAllowedBooleanAttribute

Description: Returns the attribute “allLogicalDirectionsAllowed” defined on the given element.

- Returns: attributeValue of type **Boolean** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.7 getAllLogicalInitiativesAllowedBooleanAttribute

Description: Returns the attribute “allLogicalInitiativesAllowed” defined on the given element.

- Returns: attributeValue of type **Boolean** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.8 getAlternateRegisterRefAttribute

Description: Returns the attribute “alternateRegisterRef” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.9 getAlternateRegisterRefAttributeByID

Description: Returns the handle to the alternateRegister referenced from the given alternateRegisterRef element.

- Returns: alternateRegisterID of type **String** - Handle to the referenced alternateRegister element
- Input: alternateRegisterRefID of type **String** - Handle to an alternateRegisterRef element

F.7.43.10 getAppendBooleanAttribute

Description: Returns the attribute “append” defined on the given file buildCommand flags element.

- Returns: attributeValue of type **Boolean** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.11 getArrayIdAttribute

Description: Returns the attribute “arrayId” of the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.12 getBankAlignmentAttribute

Description: Returns the attribute “bankAlignment” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.13 getBankRefAttribute

Description: Returns the attribute “bankRef” defined on the given memoryMapRef element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.14 getBankRefAttributeByID

Description: Returns the handle to the bank referenced from the given bankRef element.

- Returns: bankID of type **String** - Handle to the referenced bank element
- Input: bankRefID of type **String** - Handle to a bankRef element

F.7.43.15 getBusRefAttribute

Description: Returns the attribute “busRef” defined on the given element

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.16 getBusRefAttributeByID

Description: Returns the handle to the busInterface defined on the given element.

- Returns: busInterfaceID of type **String** - Handle to the referenced busInterface
- Input: elementID of type **String** - Handle to an element

F.7.43.17 getCellStrengthAttribute

Description: Returns the attribute “cellStrength” defined on the given cellSpecification element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.18 getChoiceRefAttribute

Description: Returns the attribute “choiceRef” defined on the given parameter element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.19 getChoiceRefAttributeByID

Description: Returns the handle to the referenced choice defined on the given parameter element.

- Returns: choiceID of type **String** - Handle of the referenced choice
- Input: parameterBaseTypeID of type **String** - Handle to a parameter element

F.7.43.20 getClockEdgeAttribute

Description: Returns the attribute “clockEdge” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.21 getClockNameAttribute

Description: Returns the attribute “clockName” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.22 getClockSourceAttribute

Description: Returns the attribute “clockSource” defined on the given otherclockDriver element.

- Returns: attributeValue of type **String** - The attribute value

- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.23 getComponentInstanceRefAttribute

Description: Returns the attribute “componentInstanceRef” defined on the given element

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.24 getComponentInstanceRefAttributeByID

Description: Returns the handle to the componentInstance referenced from the given element.

- Returns: componentInstanceID of type **String** - Handle to the referenced componentInstance element
- Input: elementID of type **String** - Handle to an element

F.7.43.25 getComponentRefAttribute

Description: Returns the attribute “componentRef” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.26 getComponentRefAttributeByID

Description: Returns the handle to the componentInstance defined on the given element.

- Returns: componentInstanceID of type **String** - Handle to the referenced componentInstance
- Input: interfaceRefID of type **String** - Handle to an interfaceRef element

F.7.43.27 getConfigGroupsAttribute

Description: Returns the attribute “configGroups” defined on the given parameter element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.28 getConstrainedAttributeValues

Description: Returns the attribute “constrained” list defined on the given wireTypeName element.

- Returns: attributeValue of type **String** - The attribute value list
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.29 getConstraintSetIdAttribute

Description: Returns the attribute “constraintSetId” defined on the given constraintSet element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.30 getCustomAttribute

Description: Returns the attribute “custom” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.31 getDataTypeAttribute

Description: Returns the attribute “dataType” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.32 getDataTypeDefinitionAttribute

Description: Returns the attribute “dataTypeDefinition” defined on the given parameter element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.33 getDefaultBooleanAttribute

Description: Returns the attribute “default” defined on the given logicalName element.

- Returns: `attributeValue` of type ***Boolean*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.34 getDelayTypeAttribute

Description: Returns the attribute “delayType” defined on the given timingConstraint element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.35 getDirectionAttribute

Description: Returns the attribute “direction” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.36 getDriverTypeAttribute

Description: Returns the attribute “driverType” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has the attribute “driverType”

F.7.43.37 getExactBooleanAttribute

Description: Returns the attribute “exact” defined on the given typeName element.

- Returns: `attributeValue` of type ***Boolean*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.38 getExternalDeclarationsBooleanAttribute

Description: Returns the attribute “externalDeclarations” defined on the given file isInclude element.

- Returns: `attributeValue` of type ***Boolean*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.39 getFieldRefAttribute

Description: Returns the attribute “fieldRef” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.40 getFieldRefAttributeByID

Description: Returns the handle to field referenced from the given fieldRef element.

- Returns: `fieldID` of type ***String*** - Handle to the referenced field element
- Input: `fieldRefID` of type ***String*** - Handle to a fieldRef element

F.7.43.41 getFiledAttribute

Description: Returns the “fileId” attribute defined on the given file element.

- Returns: `fileId` of type ***String*** - The value of the fileId attribute
- Input: `elementID` of type ***String*** - Handle to a file element

F.7.43.42 getFlowTypeAttribute

Description: Returns the attribute “flowType” defined on the given isFlowControl element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.43 getForceBooleanAttribute

Description: Returns the attribute “force” defined on the given accessHandle element.

- Returns: `attributeValue` of type ***Boolean*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.44 getGroupAttribute

Description: Returns the attribute “group” defined on the given abstractorMode element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.45 getHelpAttribute

Description: Returns the attribute “help” defined on the given choice enumeration element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.46 getHiddenBooleanAttribute

Description: Returns the attribute “hidden” defined on the given element.

- Returns: `attributeValue` of type ***Boolean*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.47 getIdAttribute

Description: Returns the attribute “id” defined on the given element

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to

F.7.43.48 getImageIdAttribute

Description: Returns the attribute “imageId” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.49 getImageTypeAttribute

Description: Returns the attribute “imageType” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.50 getImplicitBooleanAttribute

Description: Returns the attribute “implicit” defined on the given `typeName` element.

- Returns: `attributeValue` of type ***Boolean*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.51 getIndexVarAttribute

Description: Returns the attribute “indexVar” defined on the given `dim` element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.52 getInitiatorRefAttribute

Description: Returns the attribute “initiatorRef” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.53 getInitiatorRefAttributeByID

Description: Returns the handle to the initiator defined on the given element.

- Returns: `initiatorID` of type ***String*** - Handle to the referenced initiator
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.54 getInterfaceModeAttribute

Description: Returns the attribute “interfaceMode” defined on the given `monitor` element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.55 getInvertAttribute

Description: Returns the attribute “invert” defined on the given portMap element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.56 getIsIOBooleanAttribute

Description: Returns the attribute “isIO” defined on the given subPort element.

- Returns: attributeValue of type **Boolean** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.57 getLevelAttribute

Description: Returns the attribute “level” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.58 getLibextAttribute

Description: Returns the attribute “libext” defined on the given fileType element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.59 getLibraryAttribute

Description: Returns the attribute “library” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.60 getMandatoryBooleanAttribute

Description: Returns the attribute “mandatory” defined on the given extension element from payload.

- Returns: attributeValue of type **Boolean** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.61 getMaximumAttribute

Description: Returns the attribute “maximum” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.62 getMaximumDoubleAttribute

Description: Returns the attribute “maximum” defined on the given element.

- Returns: attributeValue of type **Double** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.63 getMaximumIntAttribute

Description: Returns the attribute “maximum” defined on the given element.

- Returns: `attributeValue` of type **Long** - The attribute value
- Input: `elementID` of type **String** - Handle to an element

F.7.43.64 getMemoryMapRefAttribute

Description: Returns the attribute “memoryMapRef” defined on the given element.

- Returns: `attributeValue` of type **String** - The attribute value
- Input: `elementID` of type **String** - Handle to an element that has an attribute

F.7.43.65 getMemoryMapRefAttributeByID

Description: Returns the handle to the memoryMap referenced from the given element.

- Returns: `memoryMapID` of type **String** - Handle to the referenced memoryMap
- Input: `elementID` of type **String** - Handle to an element

F.7.43.66 getMemoryRemapRefAttributeByID

Description: Returns the handle to the memoryRemap referenced from the given element.

- Returns: `memoryRemapID` of type **String** - Handle to the referenced memoryRemap
- Input: `elementID` of type **String** - Handle to an element

F.7.43.67 getMemoryRemapRefAttribute

Description: Returns the attribute “memoryRemapRef” defined on the given element.

- Returns: `attributeValue` of type **String** - The attribute value
- Input: `elementID` of type **String** - Handle to an element that has an attribute

F.7.43.68 getMinimumAttribute

Description: Returns the attribute “minimum” defined on the given element.

- Returns: `attributeValue` of type **String** - The attribute value
- Input: `elementID` of type **String** - Handle to an element

F.7.43.69 getMinimumDoubleAttribute

Description: Returns the attribute “minimum” defined on the given element.

- Returns: `attributeValue` of type **Double** - The attribute value
- Input: `elementID` of type **String** - Handle to an element

F.7.43.70 getMinimumIntAttribute

Description: Returns the attribute “minimum” defined on the given element.

- Returns: `attributeValue` of type **Long** - The attribute value
- Input: `elementID` of type **String** - Handle to an element

F.7.43.71 getMisalignmentAllowedBooleanAttribute

Description: Returns the attribute “MisalignmentAllowed” defined on the given element.

- Returns: `attributeValue` of type **Boolean** - The attribute value
- Input: `elementID` of type **String** - Handle to an element
- Input:

F.7.43.72 getModeRefAttribute

Description: Returns the attribute “modeRef” defined on the given element

- Returns: `attributeValue` of type **String** - The attribute value
- Input: `elementID` of type **String** - Handle to an element that has an attribute

F.7.43.73 getModifyAttribute

Description: Returns the attribute “modify” defined on the given element.

- Returns: `attributeValue` of type **String** - The attribute value
- Input: `elementID` of type **String** - Handle to an element

F.7.43.74 getMultipleGroupSelectionOperatorAttribute

Description: Returns the “multipleGroupSelectionOperato” attribute defined on the given groupSelector element.

- Returns: `multipleGroupSelectionOperator` of type **String** - The value of the `multipleGroupSelectionOperator` attribute
- Input: `elementID` of type **String** - Handle to a groupSelector element

F.7.43.75 getNameAttribute

Description: Returns the attribute “name” defined on the given element.

- Returns: `attributeValue` of type **String** - The attribute value
- Input: `elementID` of type **String** - Handle to an element

F.7.43.76 getOrderFloatAttribute

Description: Returns the attribute “order” defined on the given element.

- Returns: `attributeValue` of type **Float** - The attribute value
- Input: `elementID` of type **String** - Handle to an element

F.7.43.77 getOtherAnyAttribute

Description: Returns the value of the given anyAttribute name defined on the given element.

- Returns: `value` of type **String** - The attribute value
- Input: `attributeContainerID` of type **String** - Handle to an element containing the attribute
- Input: `attributeName` of type **String** - The attribute name

F.7.43.78 getOtherAttribute

Description: Returns the attribute “other” defined on the given cellFunction element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.79 getOtherAttributes

Description: Returns all the otherAttribute names defined on the given element.

- Returns: attributesName of type **String** - List of attributes names
- Input: attributeContainerID of type **String** - Handle to the container of the attributes

F.7.43.80 getPackedBooleanAttribute

Description: Returns the attribute “packed” defined on the given structured element.

- Returns: attributeValue of type **Boolean** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.81 getParameterIdAttribute

Description: Returns the attribute “parameterId” defined on the given parameter element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.82 getPathAttribute

Description: Returns the attribute “path” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.83 getPhantomBooleanAttribute

Description: Returns the attribute “phantom” defined on the given element.

- Returns: attributeValue of type **Boolean** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.84 getPortRefAttribute

Description: Returns the attribute “portRef” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.85 getPortRefAttributeByID

Description: Returns the handle to the port referenced from the given portSlice element.

- Returns: portID of type **String** - Handle to the referenced port element
- Input: portSliceIDOrInternalPortRef of type **String** - Handle to a portSlice element

F.7.43.86 getPowerDomainRefAttribute

Description: Returns the attribute “powerDomainRef” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value

- Input: elementID of type **String** - Handle to an element

F.7.43.87 getPowerDomainRefAttributeByID

Description: Returns the handle to the powerDomain referenced from the port qualifier isPowerEn element.

- Returns: powerDomainID of type **String** - Handle to the referenced powerDomain element
- Input: powerEnID of type **String** - Handle to a powerEn element

F.7.43.88 getPrefixAttribute

Description: Returns the attribute “prefix” defined on the given parameter element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.89 getPriorityIntAttribute

Description: Returns the attribute “priority” defined on the given element.

- Returns: attributeValue of type **Long** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.90 getPromptAttribute

Description: Returns the attribute “prompt” defined on the given parameter element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.91 getReferenceIdAttribute

Description: Returns the attribute “referenceId” defined on the given configurableElementValue element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.43.92 getRegisterFileRefAttribute

Description: Returns the attribute “registerFileRef” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.93 getRegisterFileRefAttributeByID

Description: Returns the handle to the registerFile referenced from the given registerFileRef element.

- Returns: registerFileRefID of type **String** - Handle to the referenced registerFile element
- Input: registerFileRefID of type **String** - Handle to a registerFileRef element

F.7.43.94 getRegisterRefAttribute

Description: Returns the attribute “registerRef” defined on the given element.

- Returns: attributeValue of type **String** - The attribute value
- Input: elementID of type **String** - Handle to an element

F.7.43.95 getReplicateBooleanAttribute

Description: Returns the attribute “replicate” defined on the given function element.

- Returns: `attributeValue` of type ***Boolean*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.96 getResetTypeRefAttribute

Description: Returns the “resetTypeRef” attribute defined on the given resetTypeReference element.

- Returns: `resetTypeRef` of type ***String*** - The value of the referenced resetType
- Input: `elementID` of type ***String*** - Handle to a resetTypeReference element

F.7.43.97 getResetTypeRefAttributeByID

Description: Returns the handle to the resetType element referenced from the given reset element.

- Returns: `resetTypeID` of type ***String*** - Handle to the referenced resetType element
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.43.98 getResolveAttribute

Description: Returns the attribute “resolve” defined on the given parameter element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.99 getScopeAttribute

Description: Returns the attribute “scope” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to a componentGenerator element

F.7.43.100 getSegmentRefAttribute

Description: Returns the attribute “segmentRef” defined on the given subSpaceMap element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.101 getSegmentRefAttributeByID

Description: Returns the handle to the segment referenced from the given subspaceMap element.

- Returns: `segmentID` of type ***String*** - Handle to the referenced segment
- Input: `subSpaceMapID` of type ***String*** - Handle to a subspaceMap element

F.7.43.102 getSignAttribute

Description: Returns the attribute “sign” defined on the given parameter element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.103 getStrictBooleanAttribute

Description: Returns the attribute “strict” defined on the given language element.

- Returns: `attributeValue` of type ***Boolean*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.104 getSubPortRefAttribute

Description: Returns the attribute “subPortRef” defined on the given subPortReference element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.105 getSubPortRefAttributeByID

Description: Returns the handle to the subPort referenced from the given subPortReference.

- Returns: `subPortID` of type ***String*** - Handle to the referenced subPort element
- Input: `subPortReferenceID` of type ***String*** - Handle to a subPortReference element

F.7.43.106 getTestConstraintAttribute

Description: Returns the attribute “testConstraint” defined on the given testable element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.107 getTextAttribute

Description: Returns the attribute “text” defined on the given choice enumeration element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.108 getTypeAttribute

Description: Returns the attribute “type” defined on the given parameter element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.109 getTypeDefinitionsAttribute

Description: Returns the attribute “typeDefinitions” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.110 getUniqueBooleanAttribute

Description: Returns the “unique” boolean attribute defined on the given generatorChainSelector element.

- Returns: `isUnique` of type ***Boolean*** - The value of the unique attribute (false if not defined)
- Input: `elementID` of type ***String*** - Handle to a generatorChainSelector element

F.7.43.111 getUnitAttribute

Description: Returns the attribute “unit” defined on the given parameter element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.112 getUnitsAttribute

Description: Returns the attribute “units” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.113 getUsageAttribute

Description: Returns the attribute “usage” defined on the given `enumeratedValue` element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.114 getUsageTypeAttribute

Description: Returns the attribute “usageType” defined on the given parameter element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.115 getUserAttribute

Description: Returns the attribute “user” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.116 getVectorIdAttribute

Description: Returns the attribute “vectorId” of the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.117 getVendorAttribute

Description: Returns the attribute “vendor” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.118 getVersionAttribute

Description: Returns the attribute “version” defined on the given element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element

F.7.43.119 getViewRefAttribute

Description: Returns the attribute “viewRef” defined on the given view element.

- Returns: `attributeValue` of type ***String*** - The attribute value
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.43.120 getViewRefAttributeByID

Description: Returns the handle to view referenced from the given viewConfiguration or view element.

- Returns: `viewID` of type ***String*** - Handle to the referenced view element
- Input: `viewConfigurationOrViewID` of type ***String*** - Handle to a viewConfiguration element or a view

F.7.44 Element attribute (EXTENDED)

F.7.44.1 addConstrainedAttribute

Description: Adds the attribute “constrained” in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a wireTypeName element
- Input: `constrained` of type ***String*** - Attribute value

F.7.44.2 isSetAttribute

Description: Checks if the given attribute is set or not.

- Returns: `isSet` of type ***Boolean*** - True if the attribute is defined; False otherwise
- Input: `elementID` of type ***String*** - Handle to the parent of the attribute to check
- Input: `attributeName` of type ***String*** - The attribute name

F.7.44.3 removeAllBitsAttribute

Description: Removes the attribute “allBits” from the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a wireTypeName element

F.7.44.4 removeAppendAttribute

Description: Removes the attribute “append” of the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a wireTypeName element

F.7.44.5 removeArrayIdAttribute

Description: Removes the attribute “arrayId” of the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element

F.7.44.6 removeAttribute

Description: Removes the given attribute name from its containing element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to an element that has an attribute
- Input: `name` of type **String** - Attribute name

F.7.44.7 removeCellStrengthAttribute

Description: Removes the attribute “cellStrength” from the given element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to a cellSpecification element

F.7.44.8 removeChoiceRefAttribute

Description: Removes the attribute “choiceRef” in the given element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to a parameter element

F.7.44.9 removeClockEdgeAttribute

Description: Removes the value of the element `clockEdge` attribute.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to the element

F.7.44.10 removeClockNameAttribute

Description: Removes the value of the element `clockName` attribute.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to the element

F.7.44.11 removeClockSourceAttribute

Description: Removes the attribute “clockSource” from the given element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to an otherclockDriver element

F.7.44.12 removeConstrainedAttribute

Description: Removes the given constrained from the attribute “constrained” list in the given element

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to a wireTypeName element
- Input: `constrained` of type **String** - Attribute expression

F.7.44.13 removeConstraintSetIdAttribute

Description: Removes the value of the element `constraintSetId` attribute.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to the element

F.7.44.14 removeDataTypeAttribute

Description: Removes the attribute “dataType” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element

F.7.44.15 removeDataTypeDefinitionAttribute

Description: Removes the attribute “dataTypeDefinition” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element

F.7.44.16 removeDefaultAttribute

Description: Removes the default attribute of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.44.17 removeDelayTypeAttribute

Description: Removes the value of the element clockEdge attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.18 removeDirectionAttribute

Description: Removes the attribute “direction” of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element

F.7.44.19 removeDriverTypeAttribute

Description: Removes the attribute “driverType” to the given expression for the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.44.20 removeExternalDeclarationsAttribute

Description: Removes the externalDeclarations attribute of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.44.21 removeFileDialogAttribute

Description: Removes the fileId attribute of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.44.22 removeFlowTypeAttribute

Description: Removes the attribute “flowType” from the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an isFlowControl element

F.7.44.23 removeForceAttribute

Description: Removes the value of the element force attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.24 removeGroupAttribute

Description: Removes the attribute “group” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to element which contains the id.

F.7.44.25 removeHelpAttribute

Description: Removes the value of the element help attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.26 removeHiddenAttribute

Description: Removes the value of the element hidden attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.27 removeIdAttribute

Description: Removes the attribute “id” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to element which contains the id.

F.7.44.28 removeImageTypeAttribute

Description: Removes the imageType attribute of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.44.29 removeImplicitAttribute

Description: Removes the attribute “implicit” of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a typeName element

F.7.44.30 removeInvertAttribute

Description: Removes the value of the element invert attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.31 removeIsIOAttribute

Description: Removes the attribute “isIO” from the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a subPort element

F.7.44.32 removeLevelAttribute

Description: Removes the attribute “level” from the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element

F.7.44.33 removeLibextAttribute

Description: Removes the libext attribute of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute

F.7.44.34 removeMaximumAttribute

Description: Removes the maximum attribute of the given numeric expression.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an expression element

F.7.44.35 removeMinimumAttribute

Description: Removes the minimum attribute of the given numeric expression.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an expression element

F.7.44.36 removeMisalignmentAllowedAttribute

Description: Removes the value of the element misalignmentAllowed attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.37 removeModeAttribute

Description: Removes the value of the element mode attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.38 removeModifyAttribute

Description: Removes the value of the element modify attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.39 removeMultipleGroupSelectionOperatorAttribute

Description: Removes the value of the element multipleGroupSelectionOperator attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a groupSelector

F.7.44.40 removeOrderAttribute

Description: Removes the attribute “order” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element

F.7.44.41 removeOtherAnyAttribute

Description: Removes the otherAny attribute name and value on the given element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: attributeContainerID of type **String** - Handle to the element containing the attribute
- Input: attributeName of type **String** - Name of the attribute

F.7.44.42 removeOtherAttribute

Description: Removes the value of the element other attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.43 removePackedAttribute

Description: Removes the attribute “packed” from the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a structured element

F.7.44.44 removeParameterIdAttribute

Description: Removes the attribute “parameterId” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element

F.7.44.45 removePathAttribute

Description: Removes the value of the element path attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to the element

F.7.44.46 removePhantomAttribute

Description: Removes the attribute “phantom” of the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element

F.7.44.47 removePowerDomainRefAttribute

Description: Removes the attribute “powerDomainRef” from the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element

F.7.44.48 removePrefixAttribute

Description: Removes the attribute “prefix” in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a parameter element

F.7.44.49 removePromptAttribute

Description: Removes the attribute “prompt” in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a parameter element

F.7.44.50 removeReplicateAttribute

Description: Removes the attribute “replicate” of the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a function element

F.7.44.51 removeResetTypeRefAttribute

Description: Removes the value of the element `resetTypeRef` attribute.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to the element

F.7.44.52 removeResolveAttribute

Description: Removes the attribute “resolve” in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a parameter element

F.7.44.53 removeScopeAttribute

Description: Removes the scope attribute of the element `componentGenerator` or `abstractorGenerator` attribute.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element

F.7.44.54 removeSegmentRefAttribute

Description: Removes the attribute “segmentRef” of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a subSpaceMap element

F.7.44.55 removeSignAttribute

Description: Removes the attribute “sign” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element

F.7.44.56 removeStrictAttribute

Description: Removes the attribute “strict” from the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a language element

F.7.44.57 removeTestConstraintAttribute

Description: Removes the attribute “testConstraint” of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a testable element

F.7.44.58 removeTextAttribute

Description: Removes the value of the element text attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element

F.7.44.59 removeTypeAttribute

Description: Removes the attribute “type” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to parameter element

F.7.44.60 removeUniqueAttribute

Description: Removes the value of the element unique attribute.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element

F.7.44.61 removeUnitAttribute

Description: Removes the attribute “unit” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element

F.7.44.62 removeUnitsAttribute

Description: Removes the attribute “units” from the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a `clockPeriod` or `clockPulseOffset` or `clockPulseDuration` element

F.7.44.63 removeUsageTypeAttribute

Description: Removes the attribute “usageType” in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a parameter element

F.7.44.64 removeUserAttribute

Description: Removes the user attribute of the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute

F.7.44.65 removeVectorIdAttribute

Description: Removes the attribute “vectorId” of the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element

F.7.44.66 setAddressBlockRefAttribute

Description: Sets the attribute “addressBlockRef” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an `addressBlockRef` element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.67 setAddressSpaceRefAttribute

Description: Sets the attribute “addressSpaceRef” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.68 setAllBitsBooleanAttribute

Description: Sets the attribute “allBits” to the given expression in the element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a `wirePort width` element
- Input: `value` of type ***Boolean*** - Attribute value

F.7.44.69 setAllLogicalDirectionsAllowedBooleanAttribute

Description: Sets the attribute “allLogicalDirectionsAllowed” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: expression of type **Boolean** - Attribute expression

F.7.44.70 setAllLogicalInitiativesAllowedBooleanAttribute

Description: Sets the attribute “allLogicalInitiativesAllowed” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: expression of type **Boolean** - Attribute expression

F.7.44.71 setAlternateRegisterRefAttribute

Description: Sets the attribute “alternateRegisterRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.72 setAppendBooleanAttribute

Description: Sets the attribute “append” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a file buildCommand flags element
- Input: append of type **Boolean** - Attribute value

F.7.44.73 setArrayIdAttribute

Description: Sets the attribute “arrayId” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: arrayId of type **String** - Attribute value

F.7.44.74 setBankAlignmentAttribute

Description: Sets the attribute “bankAlignment” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.75 setBankRefAttribute

Description: Sets the attribute “bankRef” to the given expression in the given memoryMapRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: expression of type **String** - Attribute expression

F.7.44.76 setBusRefAttribute

Description: Sets the attribute “busRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.77 setCellStrengthAttribute

Description: Sets the attribute “cellStrength” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a cellSpecification element
- Input: expression of type **String** - Attribute expression

F.7.44.78 setChoiceRefAttribute

Description: Sets the attribute “choiceRef” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element
- Input: expression of type **String** - Attribute expression

F.7.44.79 setClockEdgeAttribute

Description: Sets the attribute “clockEdge” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.80 setClockNameAttribute

Description: Sets the attribute “clockName” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.81 setClockSourceAttribute

Description: Sets the attribute “clockSource” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an otherClockDriver element
- Input: expression of type **String** - Attribute expression

F.7.44.82 setComponentInstanceRefAttribute

Description: Sets the attribute “componentInstanceRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.83 setComponentRefAttribute

Description: Sets the attribute “componentRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.84 setConfigGroupsAttribute

Description: Sets the attribute “configGroups” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element
- Input: configGroups of type **String** - Attribute value

F.7.44.85 setConstraintSetIdAttribute

Description: Sets the attribute “constraintSetId” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a constraintSet element
- Input: expression of type **String** - Attribute expression

F.7.44.86 setCustomAttribute

Description: Sets the attribute “custom” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.87 setDataTypeAttribute

Description: Sets the attribute “dataType” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.88 setDataTypeDefinitionAttribute

Description: Sets the attribute “dataTypeDefinition” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element
- Input: expression of type **String** - Attribute expression

F.7.44.89 setDefaultBooleanAttribute

Description: Sets the attribute “default” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a file logicalName element
- Input: defaultValue of type **Boolean** - Attribute value

F.7.44.90 setDelayTypeAttribute

Description: Sets the attribute “delayType” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a timingConstraint element
- Input: expression of type **String** - Attribute expression

F.7.44.91 setDirectionAttribute

Description: Sets the attribute “direction” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: direction of type **String** - Attribute value

F.7.44.92 setDriverTypeAttribute

Description: Sets the attribute “driverType” to the given expression for the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: driverType of type **String** - The driver type value

F.7.44.93 setExactBooleanAttribute

Description: Sets the attribute “exact” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a typeName element
- Input: exact of type **Boolean** - Attribute value

F.7.44.94 setExternalDeclarationsBooleanAttribute

Description: Sets the attribute “externalDeclarations” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a file isInclude element
- Input: externalDeclarations of type **Boolean** - Attribute value

F.7.44.95 setFieldRefAttribute

Description: Sets the attribute “fieldRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.96 setFileIdAttribute

Description: Sets the attribute “fileId” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a file buildCommand flags element
- Input: expression of type **String** - Attribute expression

F.7.44.97 setFlowTypeAttribute

Description: Sets the attribute “flowType” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an isFlowControl element
- Input: expression of type **String** - Attribute expression

F.7.44.98 setForceBooleanAttribute

Description: Sets the attribute “force” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an accessHandle element
- Input: force of type **Boolean** - Attribute expression

F.7.44.99 setGroupAttribute

Description: Sets the attribute “group” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an abstractorMode element
- Input: expression of type **String** - Attribute expression

F.7.44.100 setHelpAttribute

Description: Sets the attribute “help” to the given value for the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: help of type **String** - ChoiceEnumeration help

F.7.44.101 setHiddenBooleanAttribute

Description: Sets the attribute “hidden” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: hidden of type **Boolean** - True if the given element is hidden

F.7.44.102 setIdAttribute

Description: Sets the attribute “id” to the given value in the given element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: id of type **String** - Attribute value

F.7.44.103 setImageIdAttribute

Description: Sets the attribute “imageId” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: imageId of type **String** - Attribute value

F.7.44.104 setImageTypeAttribute

Description: Sets the attribute “imageType” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: imageType of type **String** - Attribute value

F.7.44.105 setImplicitBooleanAttribute

Description: Sets the attribute “implicit” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a typeName element
- Input: implicit of type **Boolean** - Attribute value

F.7.44.106 setIndexVarAttribute

Description: Sets the attribute “indexVar” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a dim element
- Input: expression of type **String** - Attribute expression

F.7.44.107 setInitiatorRefAttribute

Description: Sets the attribute “initiatorRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.108 setInterfaceModeAttribute

Description: Sets the attribute “interfaceMode” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a monitor element
- Input: expression of type **String** - Attribute expression

F.7.44.109 setInvertAttribute

Description: Sets the attribute “invert” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a portMap element
- Input: expression of type **String** - Attribute expression

F.7.44.110 setIsIOBooleanAttribute

Description: Sets the attribute “isIO” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a subPort element
- Input: value of type **Boolean** - Attribute value

F.7.44.111 setLevelAttribute

Description: Sets the attribute “level” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.112 setLibextAttribute

Description: Sets the attribute “libext” to the given expression in the given fileType element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: expression of type **String** - Attribute expression

F.7.44.113 setLibraryAttribute

Description: Sets the attribute “library” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.114 setMandatoryBooleanAttribute

Description: Sets the attribute “mandatory” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a payload extension element
- Input: expression of type **Boolean** - Attribute boolean value

F.7.44.115 setMaximumAttribute

Description: Sets the attribute “maximum” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: value of type **String** - Attribute value

F.7.44.116 setMaximumDoubleAttribute

Description: Sets the attribute “maximum” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: value of type **Double** - Attribute value

F.7.44.117 setMaximumIntAttribute

Description: Sets the attribute “maximum” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: value of type **Long** - Attribute value

F.7.44.118 setMemoryMapRefAttribute

Description: Sets the attribute “memoryMapRef” to the given expression in the given memoryMapRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: expression of type **String** - Attribute expression

F.7.44.119 setMemoryRemapRefAttribute

Description: Sets the attribute “memoryRemapRef” to the given expression in the given memoryMapRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has an attribute
- Input: expression of type **String** - Attribute expression

F.7.44.120 setMinimumAttribute

Description: Sets the attribute “minimum” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: value of type **String** - Attribute value

F.7.44.121 setMinimumDoubleAttribute

Description: Sets the attribute “minimum” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: value of type **Double** - Attribute value

F.7.44.122 setMinimumIntAttribute

Description: Sets the attribute “minimum” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: value of type **Long** - Attribute value

F.7.44.123 setMisalignmentAllowedBooleanAttribute

Description: Sets the attribute “misalignmentAllowed” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: misalignmentAllowed of type **Boolean** - True if the given element is misalignmentAllowed

F.7.44.124 setModeRefAttribute

Description: Sets the attribute “modeRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: elementID of type **String** - Handle to a view element
- Input: expression of type **String** - Attribute expression

F.7.44.125 setModifyAttribute

Description: Sets the attribute “modify” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.126 setMultipleGroupSelectionOperatorAttribute

Description: Sets the attribute “multipleGroupSelectionOperator” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a groupSelector element
- Input: multipleGroupSelectionOperator of type **String** - Specifies the OR or AND selection operator if there is more than one group name

F.7.44.127 setNameAttribute

Description: Sets the attribute “name” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.128 setOrderFloatAttribute

Description: Sets the attribute “order” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element
- Input: order of type **Double** - Attribute value

F.7.44.129 setOtherAnyAttribute

Description: Sets the otherAny attribute name and value on the given element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: attributeContainerID of type **String** - Handle to the element containing the attribute
- Input: attributeName of type **String** - Name of the attribute
- Input: value of type **String** - Value of the attribute

F.7.44.130 setOtherAttribute

Description: Sets the attribute “other” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a cellFunction element
- Input: expression of type **String** - Attribute expression

F.7.44.131 setPackedBooleanAttribute

Description: Sets the attribute “packed” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a structured element
- Input: `packed` of type ***Boolean*** - Attribute value

F.7.44.132 setParameterIdAttribute

Description: Sets the attribute “parameterId” to the given value in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a parameter element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.133 setPathAttribute

Description: Sets the attribute “path” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.134 setPhantomBooleanAttribute

Description: Sets the attribute “phantom” to the given value in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element
- Input: `hidden` of type ***Boolean*** - True if the given element is hidden

F.7.44.135 setPortRefAttribute

Description: Sets the attribute “portRef” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.136 setPowerDomainRefAttribute

Description: Sets the attribute “powerDomainRef” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.137 setPrefixAttribute

Description: Sets the attribute “prefix” to the given value in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a parameter element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.138 setPriorityIntAttribute

Description: Sets the attribute “priority” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a modeRef element
- Input: expression of type **Long** - Attribute expression

F.7.44.139 setPromptAttribute

Description: Sets the attribute “prompt” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element
- Input: expression of type **String** - Attribute expression

F.7.44.140 setReferenceldAttribute

Description: Sets the attribute “referenceId” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a configurableElementValue element
- Input: expression of type **String** - Attribute expression

F.7.44.141 setRegisterFileRefAttribute

Description: Sets the attribute “registerFileRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.142 setRegisterRefAttribute

Description: Sets the attribute “registerRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.143 setReplicateBooleanAttribute

Description: Sets the attribute “replicate” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a function element
- Input: replicate of type **Boolean** - Attribute value

F.7.44.144 setResetTypeRefAttribute

Description: Sets the attribute “resetTypeRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.145 setResolveAttribute

Description: Sets the attribute “resolve” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element
- Input: expression of type **String** - Attribute expression

F.7.44.146 setScopeAttribute

Description: Sets the attribute “scope” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a componentGenerator element
- Input: scope of type **String** - The scope attribute applies to component generators and specifies whether the generator should be run for each instance of the entity (or module) or just once for all instances of the entity.

F.7.44.147 setSegmentRefAttribute

Description: Sets the attribute “segmentRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a subSpaceMap element
- Input: expression of type **String** - Attribute expression

F.7.44.148 setSignAttribute

Description: Sets the attribute “sign” to the given value in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element
- Input: expression of type **String** - Attribute expression

F.7.44.149 setStrictBooleanAttribute

Description: Sets the attribute “strict” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a language element
- Input: strict of type **Boolean** - Attribute expression

F.7.44.150 setSubPortRefAttribute

Description: Sets the attribute “subPortRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a subPortReference element
- Input: expression of type **String** - Attribute expression

F.7.44.151 setTestConstraintAttribute

Description: Sets the attribute “testConstraint” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a testable element

- Input: `testConstraint` of type ***String*** - Attribute value

F.7.44.152 setTextAttribute

Description: Sets the attribute “text” to the given value for the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element
- Input: `text` of type ***String*** - ChoiceEnumeration text

F.7.44.153 setTypeAttribute

Description: Sets the attribute “type” to the given value in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to parameter element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.154 setTypeDefinitionsAttribute

Description: Sets the attribute “typeDefinitions” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.155 setUniqueBooleanAttribute

Description: Sets the attribute “unique” to the given value in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a generatorChainSelector element
- Input: `unique` of type ***Boolean*** - True if the given element is unique

F.7.44.156 setUnitAttribute

Description: Sets the attribute “unit” to the given value in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a parameter element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.157 setUnitsAttribute

Description: Sets the attribute “units” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to a clockPeriod or clockPulseOffset or clockPulseDuration element
- Input: `expression` of type ***String*** - Attribute expression

F.7.44.158 setUsageAttribute

Description: Sets the attribute “usage” to the given expression in the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

- Input: elementID of type **String** - Handle to an enumeratedValue element
- Input: expression of type **String** - Attribute expression

F.7.44.159 setUsageTypeAttribute

Description: Sets the attribute “usageType” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a parameter element
- Input: expression of type **String** - Attribute expression

F.7.44.160 setUserAttribute

Description: Sets the attribute “user” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an isFlowControl or isUser or fileType element
- Input: expression of type **String** - Attribute expression

F.7.44.161 setVectorIdAttribute

Description: Sets the attribute “vectorId” in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: vectorId of type **String** - Attribute value

F.7.44.162 setVendorAttribute

Description: Sets the attribute “vendor” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.163 setVersionAttribute

Description: Sets the attribute “version” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element
- Input: expression of type **String** - Attribute expression

F.7.44.164 setViewRefAttribute

Description: Sets the attribute “viewRef” to the given expression in the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a view element
- Input: expression of type **String** - Attribute expression

F.7.45 File builder (BASE)

F.7.45.1 getBuildCommandCommand

Description: Returns the command defined on the given buildCommand element.

- Returns: commandValue of type **String** - The command value
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.45.2 getBuildCommandCommandExpression

Description: Returns the command expression defined on the given buildCommand element.

- Returns: baseAddressExpression of type **String** - The command expression
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.45.3 getBuildCommandCommandID

Description: Returns the handle to the command defined on the given buildCommand element.

- Returns: commandID of type **String** - Handle to the command element
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.45.4 getBuildCommandFlags

Description: Returns the flags defined on the given buildCommand element.

- Returns: flagsValue of type **String** - The buildCommand flags
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.45.5 getBuildCommandFlagsExpression

Description: Returns the flags expression defined on the given buildCommand element.

- Returns: flagsExpression of type **String** - The buildCommand flags expression
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.45.6 getBuildCommandFlagsID

Description: Returns the handle to the flags defined on the given buildCommand element.

- Returns: FlagsID of type **String** - Handle to the flags element
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.45.7 getBuildCommandReplaceDefaultFlags

Description: Returns the replaceDefaultFlags value defined on the given buildCommand element.

- Returns: flagsValue of type **Boolean** - The buildCommands replaceDefaultFlags value
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.45.8 getBuildCommandReplaceDefaultFlagsID

Description: Returns the handle to the replaceDefaultFlags defined on the given buildCommand element.

- Returns: replaceDefaultFlagsID of type **String** - Handle to the replaceDefaultFlags element
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.45.9 getBuildCommandTargetName

Description: Returns the targetName defined on the given buildCommand element.

- Returns: `targetName` of type ***String*** - The buildCommands target name
- Input: `buildCommandID` of type ***String*** - Handle to a buildCommand element

F.7.45.10 getBuildCommandTargetNameExpression

Description: Returns the handle to the targetName defined on the given buildCommand element.

- Returns: `targetNameID` of type ***String*** - Handle to the targetName element
- Input: `buildCommandID` of type ***String*** - Handle to a buildCommand element

F.7.45.11 getExecutableImageFileBuilderIDs

Description: Returns the handles to all the fileBuilders defined on the given executableImage element.

- Returns: `fileBuilderIDs` of type ***String*** - List of handles to the fileBuilder elements
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.45.12 getExecutableImageFileSetRefIDs

Description: Returns the handles to all the fileSetRefs defined on the given executableImage element.

- Returns: `fileSetRefIDs` of type ***String*** - List of handles to the fileSetRef elements
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.45.13 getExecutableImageLanguageToolsID

Description: Returns the handle to the languageTools defined on the given executableImage element.

- Returns: `languageToolsID` of type ***String*** - Handle to the languageTools element
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.45.14 getExecutableImageLinker

Description: Returns the linker defined on the given executableImage element.

- Returns: `linker` of type ***String*** - The linker value
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.45.15 getExecutableImageLinkerCommandFileID

Description: Returns the handle to the linkerCommandFile defined on the given executableImage element.

- Returns: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile element
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.45.16 getExecutableImageLinkerExpression

Description: Returns the linker expression defined on the given executableImage element.

- Returns: `linkerExpression` of type ***String*** - The linker expression
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.45.17 getExecutableImageLinkerFlags

Description: Returns the linkerFlags defined on the given executableImage element.

- Returns: `linkerFlags` of type ***String*** - The linker flags
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.45.18 getExecutableImageLinkerFlagsExpression

Description: Returns the linkerFlags expression defined on the given executableImage element.

- Returns: `linkerFlagsExpression` of type ***String*** - The linkerFlags expression
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.45.19 getFileBuildCommandID

Description: Returns the handle to the buildCommand defined on the given file element.

- Returns: `buildCommandID` of type ***String*** - Handle to the buildCommand element
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.45.20 getFileBuilderCommand

Description: Returns the command element defined on the given fileBuilder element.

- Returns: `command` of type ***String*** - The command value
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.21 getFileBuilderCommandExpression

Description: Returns the command expression defined on the given fileBuilder element.

- Returns: `commandExpression` of type ***String*** - The command expression
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.22 getFileBuilderCommandID

Description: Returns the handle to the command defined on the given fileBuilder element.

- Returns: `CommandID` of type ***String*** - Handle to the command
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.23 getFileBuilderFileType

Description: Returns the fileType element defined on the given fileBuilder element.

- Returns: `fileType` of type ***String*** - The fileType value
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.24 getFileBuilderFileTypeID

Description: Returns the handle to the fileType defined on the given fileBuilder element.

- Returns: `fileTypeID` of type ***String*** - Handle to the fileType element
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.25 getFileBuilderFlags

Description: Returns the flags element defined on the given fileBuilder element.

- Returns: `flags` of type ***String*** - The flags value
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.26 getFileBuilderFlagsExpression

Description: Returns the flags expression defined on the given fileBuilder element.

- Returns: `flagsExpression` of type ***String*** - The flags expression
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.27 getFileBuilderFlagsID

Description: Returns the handle to the flags defined on the given fileBuilder element.

- Returns: `FlagsID` of type ***String*** - Handle to flags element
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.28 getFileBuilderReplaceDefaultFlags

Description: Returns the replaceDefaultFlags element defined on the given fileBuilder element.

- Returns: `value` of type ***Boolean*** - The replaceDefaultFlags value
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.29 getFileBuilderReplaceDefaultFlagsExpression

Description: Returns the replaceDefaultFlags expression defined on the given fileBuilder element.

- Returns: `valueExpression` of type ***String*** - The replaceDefaultFlags expression
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.30 getFileBuilderReplaceDefaultFlagsID

Description: Returns the handle to the replaceDefaultFlags defined on the given fileBuilder element.

- Returns: `ReplaceDefaultFlagsID` of type ***String*** - Handle to the replaceDefaultFlags element
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.45.31 getGeneratorRefByID

Description: Returns the handle to the generator referenced from the given generatorRef element.

- Returns: `generatorID` of type ***String*** - Handle to the referenced generator element
- Input: `generatorRefID` of type ***String*** - Handle to a generatorRef element

F.7.45.32 getLanguageToolsFileBuilderIDs

Description: Returns the handles to all the fileBuilders defined on the given languageTools element.

- Returns: `fileBuilderIDs` of type ***String*** - List of handles to the fileBuilder elements
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.45.33 getLanguageToolsLinker

Description: Returns the linker value defined on the given languageTools element.

- Returns: `linker` of type ***String*** - The linker value
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.45.34 getLanguageToolsLinkerCommandFileID

Description: Returns the handle to the linkerCommandFile defined on the given languageTools element.

- Returns: `linkerCommandFileID` of type ***String*** - Handle to the linkerCommandFile element
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.45.35 getLanguageToolsLinkerExpression

Description: Returns the linker expression defined on the given languageTools element.

- Returns: `linker` of type ***String*** - The linker expression
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.45.36 getLanguageToolsLinkerFlags

Description: Returns the linkerFlags value defined on the given languageTools element.

- Returns: `linkerFlags` of type ***String*** - The linkerFlags value on a languageTools element
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.45.37 getLanguageToolsLinkerFlagsExpression

Description: Returns the linkerFlags expression defined on the given languageTools element.

- Returns: `linkerFlags` of type ***String*** - The linkerFlags expression on a languageTools element
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.45.38 getLanguageToolsLinkerFlagsID

Description: Returns the handle to the linkerFlags defined on the given languageTools element.

- Returns: `linkerFlagsID` of type ***String*** - Handle to the linkerFlags element
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.45.39 getLanguageToolsLinkerID

Description: Returns the handle to the linker defined on the given languageTools element.

- Returns: `linkerID` of type ***String*** - Handle to the linker element
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.45.40 getLinkerCommandFileCommandLineSwitch

Description: Returns the commandLineSwitch value defined on the given linkerCommandFile element.

- Returns: `commandLineSwitch` of type ***String*** - The commandLineSwitch value
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile element

F.7.45.41 getLinkerCommandFileCommandLineSwitchExpression

Description: Returns the commandLineSwitch expression defined on the given linkerCommandFile element.

- Returns: `commandLineSwitch` of type ***String*** - The commandLineSwitch expression
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile element

F.7.45.42 getLinkerCommandFileCommandLineSwitchID

Description: Returns the handle to the commandLineSwitch defined on the given linkerCommandFile element.

- Returns: `commandLineSwitchID` of type ***String*** - Handle to the commandLineSwitch element
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile element

F.7.45.43 getLinkerCommandFileEnable

Description: Returns the enable value defined on the given linkerCommandFile element.

- Returns: `value` of type ***Boolean*** - The enable value
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile element

F.7.45.44 getLinkerCommandFileEnableExpression

Description: Returns the enable expression defined on the given linkerCommandFile.

- Returns: `valueExpression` of type ***String*** - The enable expression
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile

F.7.45.45 getLinkerCommandFileEnableID

Description: Returns the handle to the enable element defined on the given linkerCommandFile element.

- Returns: `enableID` of type ***String*** - Handle to the enable element
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile element

F.7.45.46 getLinkerCommandFileGeneratorRefByID

Description: Returns the generatorID defined on the given linkerCommandFile element.

- Returns: `generatorID` of type ***String*** - The referenced generator
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile element
- Input: `generatorRef` of type ***String*** - The value of the generatorRef element

F.7.45.47 getLinkerCommandFileGeneratorRefByNames

Description: Returns all the generatorRefs defined on the given linkerCommandFile element.

- Returns: `generatorRefs` of type ***String*** - List of the referenced generators
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile element

F.7.45.48 getLinkerCommandFileGeneratorRefIDs

Description: Returns the handles to all the generatorRefs defined on the given linkerCommandFile element.

- Returns: `generatorRefIDs` of type ***String*** - List of handles to the generatorRef elements

- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element

F.7.45.49 getLinkerCommandFileLineSwitch

Description: Returns the commandLineSwitch value defined on the given linkerCommandFile element.

- Returns: value of type **String** - The commandLineSwitch value
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element

F.7.45.50 getLinkerCommandFileLineSwitchExpression

Description: Returns the commandLineSwitch expression defined on the given linkerCommandFile.

- Returns: valueExpression of type **String** - The commandLineSwitch expression
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile

F.7.45.51 getLinkerCommandFileName

Description: Returns the fileName defined on the given linkerCommandFile element.

- Returns: fileName of type **String** - The file name
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element

F.7.45.52 getLinkerCommandFileNameExpression

Description: Returns the fileName expression defined on the given linkerCommandFile.

- Returns: fileNameExpression of type **String** - The fileName expression
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile

F.7.45.53 getLinkerCommandFileNameID

Description: Returns the handle to the name defined on the given linkerCommandFile element.

- Returns: nameID of type **String** - Handle to the name element
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element

F.7.45.54 getLinkerCommandFileNameValue

Description: Returns the name defined on the given linkerCommandFile element.

- Returns: name of type **String** - The linkerCommandFile name
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element

F.7.46 File builder (EXTENDED)

F.7.46.1 addExecutableImageFileBuilderID

Description: Adds a fileBuilder with the given fileType and command to the given executableImage element.

- Returns: fileBuilderID of type **String** - Handle to a new fileBuilder element
- Input: executableImageID of type **String** - Handle to an executableImage element
- Input: fileType of type **String** - The fileBuilder fileType
- Input: command of type **String** - The fileBuilder commands

F.7.46.2 addExecutableImageLinkerCommandFile

Description: Adds the given linkerCommandFile name with its switch and enable flags to the given executableImage element.

- Returns: linkerCommandFileID of type **String** - Handle to a new linkerCommandFile element
- Input: executableImageID of type **String** - Handle to an executableImage element
- Input: name of type **String** - ExecutableImage linkerCommandFile
- Input: commandLineSwitch of type **String** - Flag on the command line specifying the linker command file
- Input: enable of type **Boolean** - Indicates whether to use this linker command file in the default scenario

F.7.46.3 addLanguageToolsFileBuilder

Description: Adds a fileBuilder on a languageTools element.

- Returns: fileBuilderID of type **String** - the fileBuilder identifier added on the languageTools element
- Input: languageToolsID of type **String** - Handle to a languageTools element
- Input: fileType of type **String** - The fileBuilder fileType
- Input: command of type **String** - The fileBuilder commands

F.7.46.4 addLinkerCommandFileGeneratorRef

Description: Adds a generatorRef on a linkerCommandFile element.

- Returns: generatorRefID of type **String** - the generatorRef identifier
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element
- Input: generatorRef of type **String** - The generatorRef to be added

F.7.46.5 removeBuildCommandCommand

Description: Removes a command with from the a build command element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.46.6 removeBuildCommandFlags

Description: Removes buildCommandFlags from the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element

F.7.46.7 removeBuildCommandReplaceDefaultFlags

Description: Removes ReplaceDefaultFlags from the given buildCommand element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.46.8 removeBuildCommandTargetName

Description: Removes buildCommandReplaceDefaultFlags from the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element

F.7.46.9 removeDefaultFileBuilderCommand

Description: Removes command from the given defaultFileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: defaultFileBuilderID of type **String** - Handle to a defaultFileBuilder element

F.7.46.10 removeExecutableImageFileBuilderID

Description: Removes the given fileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element

F.7.46.11 removeExecutableImageLanguageTools

Description: Removes the languageTools structure on an executableImage element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: executableImageID of type **String** - Handle to an executableImage element

F.7.46.12 removeExecutableImageLinkerCommandFile

Description: Removes the given linkerCommandFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile

F.7.46.13 removeFileBuilderCommand

Description: Removes command from the given fileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element

F.7.46.14 removeFileBuilderFlags

Description: Removes flags from the given fileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element

F.7.46.15 removeFileBuilderReplaceDefaultFlags

Description: Removes replaceDefaultFlags with the given value for the given fileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element

F.7.46.16 removeLanguageToolsFileBuilder

Description: Removes the given fileBuilder from its languageTools element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.46.17 removeLanguageToolsLinkerCommandFile

Description: Removes the linkerCommandFile structure on a languageTools element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.46.18 removeLanguageToolsLinkerFlags

Description: Removes the linkerFlag structure on a languageTools element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `languageToolsID` of type ***String*** - Handle to a languageTools element

F.7.46.19 removeLinkerCommandFileGeneratorRef

Description: Removes the given generatorRef from its containing linkerCommandFile element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorRefID` of type ***String*** - Handle of a generateRef element

F.7.46.20 setBuildCommandCommand

Description: Sets command with the given value for the given buildCommand element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `buildCommandID` of type ***String*** - Handle to a buildCommand element
- Input: `command` of type ***String*** - File buildCommand command expression

F.7.46.21 setBuildCommandFlags

Description: Sets buildCommandFlags with the given value for the given file element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `buildCommandID` of type ***String*** - Handle to a file element
- Input: `flags` of type ***String*** - File buildCommand flags

F.7.46.22 setBuildCommandReplaceDefaultFlags

Description: Sets ReplaceDefaultFlags with the given value for the given buildCommand element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `buildCommandID` of type ***String*** - Handle to a buildCommand element
- Input: `valueExpression` of type ***String*** - File buildCommand replace default flags

F.7.46.23 setBuildCommandTargetName

Description: Sets buildCommandReplaceDefaultFlags with the given value for the given file element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `buildCommandID` of type ***String*** - Handle to a file element
- Input: `targetName` of type ***String*** - File buildCommand target name

F.7.46.24 setExecutableImageLanguageTools

Description: Sets the languageTools structure on an executableImage element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: executableImageID of type **String** - Handle to an executableImage element

F.7.46.25 setExecutableImageLinker

Description: Sets the linker with the given value for the given executableImage element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: executableImageID of type **String** - Handle to an executableImage element
- Input: linker of type **String** - The executableImage linker

F.7.46.26 setExecutableImageLinkerFlags

Description: Sets the linkerFlags with the given value for the given executableImage element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: executableImageID of type **String** - Handle to an executableImage element
- Input: linkerFlags of type **String** - The executableImage linkerFlags

F.7.46.27 setFileBuildCommand

Description: Sets buildCommand with the given value for the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element

F.7.46.28 setFileBuilderCommand

Description: Sets command with the given value for the given fileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element
- Input: command of type **String** - The fileBuilder command

F.7.46.29 setFileBuilderFileType

Description: Sets fileType with the given value for the given fileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element
- Input: fileTypeValue of type **String** - The fileTypeValue

F.7.46.30 setFileBuilderFlags

Description: Sets flags with the given value for the given fileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element
- Input: flags of type **String** - The fileBuilder flags

F.7.46.31 setFileBuilderReplaceDefaultFlags

Description: Sets replaceDefaultFlags with the given value for the given fileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element
- Input: valueExpression of type **String** - The fileBuilder replace default flags

F.7.46.32 setLanguageToolsLinker

Description: Sets the linker stringExpression on a languageTools element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: languageToolsID of type **String** - Handle to a languageTools element
- Input: linker of type **String** - The linker expression to set

F.7.46.33 setLanguageToolsLinkerCommandFile

Description: Sets the linkerCommandFile structure on a languageTools element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: languageToolsID of type **String** - Handle to a languageTools element
- Input: name of type **String** - The name expression to set
- Input: commandLineSwitch of type **String** - The commandLineSwitch expression to set
- Input: enable of type **String** - The enable expression to set
- Input: linker of type **String** - The linker expression to set

F.7.46.34 setLanguageToolsLinkerFlags

Description: Sets the linkerFlags stringExpression on a languageTools element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: languageToolsID of type **String** - Handle to a languageTools element
- Input: linkerFlags of type **String** - The linkerFlags expression to set
- Input: linker of type **String** - The linker expression to set

F.7.46.35 setLinkerCommandFileCommandLineSwitch

Description: Sets the commandLineSwitch expression on a linkerCommandFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element
- Input: commandLineSwitch of type **String** - the commandLineSwitch expression to set

F.7.46.36 setLinkerCommandFileEnable

Description: Sets the enable expression on a linkerCommandFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element
- Input: enable of type **String** - the enable expression to set

F.7.46.37 setLinkerCommandFileName

Description: Sets the name of the linkerCommandFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: linkerCommandFileID of type **String** - Handle to a linkerCommandFile element
- Input: name of type **String** - File name

F.7.47 File set (BASE)

F.7.47.1 getBuildCommandReplaceDefaultFlagsExpression

Description: Returns the replaceDefaultFlags expression defined on the given buildCommand element.

- Returns: flagsExpression of type **String** - The buildCommands replaceDefaultFlags expression
- Input: buildCommandID of type **String** - Handle to a buildCommand element

F.7.47.2 getFileDefineIDs

Description: Returns the handles to all the fileDefines defined on the given file element.

- Returns: fileDefineIDs of type **String** - List of handles to the fileDefine elements
- Input: fileID of type **String** - Handle to a file element

F.7.47.3 getFileDefineSymbolValue

Description: Returns the value defined on the given fileDefine element.

- Returns: value of type **String** - The file define value
- Input: fileDefineID of type **String** - Handle to a fileDefine element

F.7.47.4 getFileDependencyIDs

Description: Returns the handles to all the dependency elements defined on the given file element.

- Returns: dependencyIDs of type **String** - List of handles to the dependency elements
- Input: fileID of type **String** - Handle to a file element

F.7.47.5 getFileExportedNameIDs

Description: Returns the handles to all the exportedNames defined on the given file element.

- Returns: exportedNameIDs of type **String** - List of handles to the exportedName elements
- Input: fileID of type **String** - Handle to a file element

F.7.47.6 getFileExportedNames

Description: Returns the exportedNames defined on the given file element.

- Returns: exportedNames of type **String** - List of exported names
- Input: fileID of type **String** - Handle to a file element

F.7.47.7 getFileFileTypeID

Description: Returns the handles to all the fileTypes defined on the given file element.

- Returns: `fileTypeID` of type ***String*** - List of handles to the `fileType` elements
- Input: `fileID` of type ***String*** - Handle to a `file` element

F.7.47.8 getFileFileTypes

Description: Returns the `fileTypes` defined on the given `file` element.

- Returns: `fileTypes` of type ***String*** - List of `file` types
- Input: `fileID` of type ***String*** - Handle to a `file` element

F.7.47.9 getFileImageTypeIDs

Description: Returns the handles to all the `imageTypes` defined on the given `file` element.

- Returns: `imageTypeIDs` of type ***String*** - List of handles to the `imageType` elements
- Input: `fileID` of type ***String*** - Handle to a `file` element

F.7.47.10 getFileImageTypes

Description: Returns all the `imageTypes` defined on the given `file` element.

- Returns: `imageTypes` of type ***String*** - List of `image` types
- Input: `fileID` of type ***String*** - Handle to a `file` element

F.7.47.11 getFileIsIncludeFile

Description: Returns the `isIncludeFile` element defined on the given `file` element.

- Returns: `value` of type ***Boolean*** - The `isIncludeFile` value
- Input: `fileID` of type ***String*** - Handle to a `file` element

F.7.47.12 getFileIsIncludeFileID

Description: Returns the handle to the `isIncludeFile` defined on the given `file` element.

- Returns: `isIncludeFileID` of type ***String*** - Handle to the `isIncludeFile` element
- Input: `fileID` of type ***String*** - Handle to a `file` element

F.7.47.13 getFileIsStructural

Description: Returns the `isStructural` element defined on the given `file` element.

- Returns: `value` of type ***Boolean*** - File `isStructural` value
- Input: `fileID` of type ***String*** - Handle to a `file` element

F.7.47.14 getFileLogicalName

Description: Returns the `logicalName` defined on the given `file` element.

- Returns: `logicalName` of type ***String*** - The logical name
- Input: `fileID` of type ***String*** - Handle to a `file` element

F.7.47.15 getFileLogicalNameID

Description: Returns the handle to the `logicalName` defined on the given `file` element.

- Returns: `logicalNameID` of type ***String*** - Handle to the `logicalName` element

- Input: `fileID` of type ***String*** - Handle to a file element

F.7.47.16 `getFileName`

Description: Returns the name defined on the given file element.

- Returns: `name` of type ***String*** - The file name
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.47.17 `getFileSetDefaultFileBuilderIDs`

Description: Returns the handles to all the fileBuilders defined on the given fileSet element.

- Returns: `fileBuilderIDs` of type ***String*** - List of handles to the fileBuilder elements
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.47.18 `getFileSetDependencyIDs`

Description: Returns the handles to all the dependency elements defined on the given fileSet element.

- Returns: `dependencyIDs` of type ***String*** - List of handles to the fileSet elements
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.47.19 `getFileSetFileIDs`

Description: Returns the handles to all the files defined on the given fileSet element.

- Returns: `fileIDs` of type ***String*** - List of handles to file elements
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.47.20 `getFileSetFunctionIDs`

Description: Returns the handles to all the functions defined on the given fileSet element.

- Returns: `functionIDs` of type ***String*** - List of handles to the function elements
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.47.21 `getFileSetGroupFileSetRefIDs`

Description: Returns the handles to all the fileSetRefs defined on the given fileSetGroup element.

- Returns: `fileSetRefIDs` of type ***String*** - List of handles to the fileSetRef elements
- Input: `fileSetRefGroupID` of type ***String*** - Handle to a fileSetRefGroup element

F.7.47.22 `getFileSetGroupIDs`

Description: Returns the handles to all the groups defined on the given fileSet element.

- Returns: `groupID` of type ***String*** - List of handles to the group elements
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.47.23 `getFileSetGroups`

Description: Returns all the groups defined on the given fileSet element.

- Returns: `groups` of type ***String*** - List of fileSet groups
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.47.24 getFileSetRefByID

Description: Returns the handle to the fileSet referenced from the given fileSetRef element.

- Returns: `fileSetRefID` of type ***String*** - Handle to the referenced fileSet element
- Input: `fileSetRefID` of type ***String*** - Handle to a fileSetRef element

F.7.47.25 getFileSetRefGroupFileSetRefIDs

Description: Returns the handles to all the fileSetRefs defined on the given fileSetRefGroup element.

- Returns: `fileSetRefIDs` of type ***String*** - List of handles to the fileSetRef elements
- Input: `fileSetRefGroupID` of type ***String*** - Handle to a fileSetRefGroup

F.7.47.26 getFileSetRefLocalNameRefByID

Description: Returns the handle to the fileSet referenced from the given fileSetRef element.

- Returns: `fileSetID` of type ***String*** - Handle to the referenced fileSet element
- Input: `fileSetRefID` of type ***String*** - Handle to a fileSetRef element

F.7.47.27 getFunctionArgumentDataType

Description: Returns the dataType defined on the given argument element.

- Returns: `dataType` of type ***String*** - The argument data type
- Input: `argumentID` of type ***String*** - Handle to an argument element

F.7.47.28 getFunctionArgumentIDs

Description: Returns the handles to all the arguments defined on the given function element.

- Returns: `argumentIDs` of type ***String*** - List of handles to argument elements
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.29 getFunctionDisabled

Description: Returns the disabled element defined on the given function element.

- Returns: `value` of type ***Boolean*** - True if the function is disabled
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.30 getFunctionDisabledExpression

Description: Returns the disabled expression defined on the given function element.

- Returns: `valueExpression` of type ***String*** - The disabled expression
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.31 getFunctionDisabledID

Description: Returns the handle to the disabled element defined on the given fileSet function element.

- Returns: `DisabledID` of type ***String*** - Handle to the disabled element
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.32 getFunctionEntryPoint

Description: Returns the entryPoint defined on the given function element.

- Returns: `entryPoint` of type ***String*** - The function entry point
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.33 getFunctionFileID

Description: Returns the handle to the file defined on the given function element.

- Returns: `fileID` of type ***String*** - Handle to the file element
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.34 getFunctionFileRefByID

Description: Returns the fileID defined on the given function element.

- Returns: `fileID` of type ***String*** - Handle to the referenced file element
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.35 getFunctionFileRefByName

Description: Returns the fileRef defined on the given function element.

- Returns: `fileRef` of type ***String*** - The referenced file
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.36 getFunctionReplicate

Description: Returns the replicate element defined on the given function element.

- Returns: `replicate` of type ***Boolean*** - The replicate value
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.37 getFunctionReturnType

Description: Returns the returnType defined on the given function element.

- Returns: `returnType` of type ***String*** - The function return type
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.38 getFunctionSourceFileIDs

Description: Returns the handles to all the sourceFiles defined on the given function element.

- Returns: `functionSourceFileIDs` of type ***String*** - List of handles to sourceFile elements
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.47.39 getFunctionSourceFileName

Description: Returns the name defined on the given function sourceFile element.

- Returns: `name` of type ***String*** - The SourceFile name
- Input: `functionSourceFileID` of type ***String*** - Handle to a functionSourceFile element

F.7.47.40 getFunctionSourceType

Description: Returns the fileType element defined on the given functionSourceFile element.

- Returns: `type` of type ***String*** - The fileType value
- Input: `functionSourceFileID` of type ***String*** - Handle to a functionSourceFile element

F.7.47.41 getSourceFileFileType

Description: Returns the fileType value defined on the given sourceFile element.

- Returns: `fileType` of type ***String*** - The fileType value
- Input: `sourceFileID` of type ***String*** - Handle to a sourceFile element

F.7.47.42 getSourceFileFileTypeID

Description: Returns the handle to the fileType defined on the given sourceFile element.

- Returns: `fileTypeID` of type ***String*** - Handle to the fileType element
- Input: `sourceFileID` of type ***String*** - Handle to a sourceFile element

F.7.47.43 getSourceFileSourceName

Description: Returns the sourceName defined on the given sourceFile element.

- Returns: `sourceName` of type ***String*** - The sourceName value
- Input: `sourceFileID` of type ***String*** - Handle to a sourceFile element

F.7.48 File set (EXTENDED)

F.7.48.1 addFileDefine

Description: Adds fileDefine with the given name and value to the given file element.

- Returns: `fileDefineID` of type ***String*** - Handle to a new fileDefine
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `name` of type ***String*** - FileDefine name
- Input: `value` of type ***String*** - FileDefine value

F.7.48.2 addFileDependency

Description: Adds given dependency to the given file element.

- Returns: `dependencyID` of type ***String*** - Handle to a new Dependency
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `dependency` of type ***String*** - File dependency

F.7.48.3 addFileExportedName

Description: Adds an exportedName on a file element.

- Returns: `exportedNameID` of type ***String*** - the exportedName identifier
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `value` of type ***String*** - The value of the exported Name

F.7.48.4 addFileFileType

Description: Adds a fileType on a file element.

- Returns: `fileTypeID` of type ***String*** - The fileType identifier on the file element
- Input: `fileID` of type ***String*** - Handle of a file element
- Input: `fileType` of type ***String*** - The fileType value to be added

F.7.48.5 addFileImageType

Description: Adds an imageType on a file element.

- Returns: `imageTypeID` of type ***String*** - The imageType identifier
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `value` of type ***String*** - The value of the image type

F.7.48.6 addFileSetDefaultFileBuilder

Description: Adds defaultFileBuilder with the given fileType to the given file element.

- Returns: `fileBuilderID` of type ***String*** - Handle to a new fileBuilder
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `fileType` of type ***String*** - The defaultFileBuilder fileType

F.7.48.7 addFileSetDependency

Description: Adds given dependency to the given fileSet element.

- Returns: `dependencyID` of type ***String*** - Handle to a new dependency
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element
- Input: `dependency` of type ***String*** - The fileSet dependency

F.7.48.8 addFileSetFile

Description: Adds file with the given name and fileTypes to the given fileSet element.

- Returns: `fileID` of type ***String*** - Handle to a new file
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element
- Input: `name` of type ***String*** - File name
- Input: `fileTypes` of type ***String[]*** - File fileTypes

F.7.48.9 addFileSetFunction

Description: Adds function with the given fileRef to the given fileSet element.

- Returns: `functionID` of type ***String*** - Handle to a new function
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element
- Input: `fileRef` of type ***String*** - Reference to the file that contains the entry point for the function

F.7.48.10 addFileSetGroup

Description: Adds a group element on the given fileSet.

- Returns: `groupID` of type ***String*** - The group identifier
- Input: `fileSetID` of type ***String*** - Handle of a fileSet element

- Input: group of type **String** - the value to set on the Group

F.7.48.11 addFileSetRefGroupFileSetRef

Description: Adds a fileSet reference (a local file name) to a fileSetRefGroup.

- Returns: fileSetRefID of type **String** - The identifier of the added fileSetRef
- Input: fileSetRefGroupID of type **String** - Handle to the fileSetRefGroup
- Input: localName of type **String** - Name of a fileSet

F.7.48.12 addFunctionArgument

Description: Adds argument with given name and value to the given function element.

- Returns: argumentID of type **String** - Handle to a new argument
- Input: functionID of type **String** - Handle to a function element
- Input: name of type **String** - Argument name
- Input: value of type **String** - Argument value
- Input: dataType of type **String** - Argument dataType

F.7.48.13 addFunctionSourceFile

Description: Adds sourceFile with the given sourceName and fileType to the given function element.

- Returns: functionSourceFileID of type **String** - Handle to a new sourceFile
- Input: functionID of type **String** - Handle to a function element
- Input: sourceFileName of type **String** - Source file name
- Input: fileType of type **String** - Source file type

F.7.48.14 removeFileBuildCommand

Description: Removes the buildCommand from its containing file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element

F.7.48.15 removeFileDefine

Description: Removes the given fileDefine element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: defineID of type **String** - Handle to a define element

F.7.48.16 removeFileDependency

Description: Removes given dependency to the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: dependencyID of type **String** - File dependency

F.7.48.17 removeFileExportedName

Description: Removes the exported name from its containing file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: exportedNameID of type **String** - Handle to an exportedName element

F.7.48.18 removeFileFileType

Description: Removes the given fileType from its containing file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileTypeID of type **String** - Handle to a fileType element

F.7.48.19 removeFileImageType

Description: Removes the given imageType from its containing file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: imageTypeID of type **String** - Handle to an imageType element

F.7.48.20 removeFileIsIncludeFile

Description: Removes isIncludeFile from the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element

F.7.48.21 removeFileIsStructural

Description: Removes isStructural from the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element

F.7.48.22 removeFileLogicalName

Description: Removes logicalName from the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element

F.7.48.23 removeFileSetDefaultFileBuilder

Description: Removes the given defaultFileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element

F.7.48.24 removeFileSetDependency

Description: Removes the given dependency from its parent.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: dependencyID of type **String** - Handle to a dependency element

F.7.48.25 removeFileSetFile

Description: Removes the given file.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element

F.7.48.26 removeFileSetFunction

Description: Removes the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element

F.7.48.27 removeFileSetGroup

Description: Removes the given group from its containing fileSet element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: groupID of type **String** - Handle of a group element

F.7.48.28 removeFunctionArgument

Description: Removes the given argument.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: argumentID of type **String** - Handle to an argument element

F.7.48.29 removeFunctionDisabled

Description: Removes disabled from the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element

F.7.48.30 removeFunctionEntryPoint

Description: Removes entryPoint from the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element

F.7.48.31 removeFunctionReturnType

Description: Removes returnType from the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element

F.7.48.32 removeFunctionSourceFile

Description: Removes the given sourceFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionSourceFileID of type **String** - Handle to a functionSourceFile element

F.7.48.33 setFileIsIncludeFile

Description: Sets isIncludeFile with the given value for the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element
- Input: value of type **Boolean** - File inIncludeFile value

F.7.48.34 setFileIsStructural

Description: Sets isStructural with the given value for the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element
- Input: value of type **Boolean** - File isStructural value

F.7.48.35 setFileLogicalName

Description: Sets logicalName with the given value for the given file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element
- Input: logicalName of type **String** - File logicalName value

F.7.48.36 setFileName

Description: Sets the name expression on a file element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileID of type **String** - Handle to a file element
- Input: name of type **String** - Name expression

F.7.48.37 setFunctionArgumentDataType

Description: Sets datatype with the given type for the given argument element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: argumentID of type **String** - Handle to an argument element
- Input: dataType of type **String** - Argument dataType

F.7.48.38 setFunctionDisabled

Description: Sets disabled to the given value for the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element
- Input: disabledExpression of type **String** - Function disabled

F.7.48.39 setFunctionEntryPoint

Description: Sets entryPoint to the given value for the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element
- Input: entryPoint of type **String** - Function entryPoint

F.7.48.40 setFunctionFileRef

Description: Sets the fileRef value for the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element
- Input: fileRef of type **String** - fileRef value

F.7.48.41 setFunctionReplicate

Description: Sets replicate to the given value for the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element
- Input: replicate of type **Boolean** - Function replicate

F.7.48.42 setFunctionReturnType

Description: Sets returnType to the given value for the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element
- Input: returnType of type **String** - Function returnType

F.7.48.43 setFunctionSourceFileName

Description: Sets sourceFileName with the given value for the given function element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: functionID of type **String** - Handle to a function element
- Input: value of type **String** - sourceFileName

F.7.48.44 setSourceFileFileType

Description: Sets the fileType on a sourceFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: sourceFileID of type **String** - Handle to a sourceFile element
- Input: value of type **String** - Value of the fileType to set

F.7.48.45 setSourceFileSourceName

Description: Sets the sourceName on a sourceFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: sourceFileID of type **String** - Handle to a sourceFile element
- Input: sourceName of type **String** - Value of sourceName to set

F.7.49 Generator (BASE)

F.7.49.1 getAbstractorGeneratorGroup

Description: Returns the group names defined on the given abstractorGenerator element.

- Returns: groups of type **String** - The group names
- Input: abstractorGeneratorID of type **String** - Handle to an abstractorGenerator element

F.7.49.2 getAbstractorGeneratorGroupIDs

Description: Returns the handles to all the groups defined on the given abstractorGenerator element.

- Returns: groupIDs of type **String** - List of handles to the group elements
- Input: abstractorGeneratorID of type **String** - Handle to an abstractorGenerator element

F.7.49.3 getComponentGeneratorGroupIDs

Description: Returns the handles to all the groups defined on the given component generator element.

- Returns: groupIDs of type **String** - List of handles to the group elements
- Input: componentGeneratorID of type **String** - Handle to a componentGenerator element

F.7.49.4 getGenerator ApiService

Description: Returns the apiService defined on the given generator element.

- Returns: apiService of type **String** - The apiService value
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.5 getGeneratorApiTypeID

Description: Returns the handle to the apiType defined on the given generator element.

- Returns: apiTypeID of type **String** - Handle to the apiType element
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.6 getGeneratorExecutable

Description: Returns the executable defined on the given generator element.

- Returns: executable of type **String** - The generator executable file
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.7 getGeneratorGeneratorExe

Description: Returns the generatorExe value defined on the given generator element

- Returns: generatorExe of type **String** - The generatorExe value
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.8 getGeneratorGroups

Description: Returns all the groups defined on the given generator element.

- Returns: groups of type **String** - List of generator groups
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.9 getGeneratorPhase

Description: Returns the phase defined on the given generator element.

- Returns: phase of type **Double** - The phase number
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.10 getGeneratorPhaseExpression

Description: Returns the phase defined on the given generator element.

- Returns: phaseExpression of type **String** - The generator phase
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.11 getGeneratorPhaseID

Description: Returns the handle to the phase defined on the given generator element.

- Returns: phaseID of type **String** - Handle to the phase element
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.12 getGeneratorScope

Description: Returns the scope defined on the given generator element.

- Returns: scope of type **String** - The generator scope
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.13 getGeneratorTransportMethodsID

Description: Returns the handle to the transportMethods defined on the given generator element.

- Returns: transportMethodsID of type **String** - Handle to the transportMethods element
- Input: generatorID of type **String** - Handle to a generator element

F.7.49.14 getTransportMethodsTransportMethodID

Description: Returns the handle to the transportMethod defined on the given transportMethods element.

- Returns: transportMethodID of type **String** - Handle to the transportMethod element
- Input: transportMethodsID of type **String** - Handle to a transportMethods element

F.7.50 Generator (EXTENDED)

F.7.50.1 addAbstractorGeneratorGroup

Description: Adds a group on an abstractorGenerator element.

- Returns: groupID of type **String** - the group identifier
- Input: abstractorGeneratorID of type **String** - Handle to an abstractorGenerator element
- Input: value of type **String** - the value of the added group

F.7.50.2 addComponentGeneratorGroup

Description: Adds a group to a componentGenerator element.

- Returns: groupID of type **String** - the group identifier
- Input: componentGeneratorID of type **String** - Handle to a componentGenerator element
- Input: value of type **String** - value to set on the group

F.7.50.3 removeAbstractorGeneratorGroup

Description: Removes the given group from the abstractor generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: groupID of type **String** - Handle to a group element

F.7.50.4 removeComponentGeneratorGroup

Description: Removes the given group from its containing component generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: groupID of type **String** - Handle to a group element

F.7.50.5 removeGeneratorApiService

Description: Removes apiService from the given generator.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator

F.7.50.6 removeGeneratorApiType

Description: Removes apiType from the given generator.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator

F.7.50.7 removeGeneratorPhase

Description: Removes phase with the given value for the given generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator element

F.7.50.8 removeGeneratorTransportMethods

Description: Removes transportMethods from the given generator.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator

F.7.50.9 removeTransportMethodsTransportMethod

Description: Removes the transportMethod identifier on a transportMethods element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transportMethodsID of type **String** - Handle to a transportMethods element

F.7.50.10 setGeneratorApiService

Description: Sets apiService with the given value for the given generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator element
- Input: apiService of type **String** - Generator apiService

F.7.50.11 setGeneratorApiType

Description: Sets apiType with the given value for the given generator | componentGenerator | abstractorGenerator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator | componentGenerator | abstractorGenerator element
- Input: apiType of type **String** - Generator apiType

F.7.50.12 setGeneratorGeneratorExe

Description: Sets the generatorExe value on a generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator element
- Input: generatorExe of type **String** - the value of the generatorExe

F.7.50.13 setGeneratorPhase

Description: Sets phase with the given value for the given generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator element
- Input: phaseExpression of type **String** - Generator phase

F.7.50.14 setGeneratorScope

Description: Sets scope with the given value for the given generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator element
- Input: scope of type **String** - Generator scope

F.7.50.15 setGeneratorTransportMethods

Description: Sets transportMethods with the given value for the given generator element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator element
- Input: transportMethod of type **String** - Generator transportMethod

F.7.50.16 setTransportMethodsTransportMethod

Description: Set the transportMethod on a transportMethods element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transportMethodsID of type **String** - Handle to a transportMethods element
- Input: transportMethod of type **String** - the transportMethod value

F.7.51 Generator chain (BASE)

F.7.51.1 getComponentGeneratorSelectorGroupSelectorID

Description: Returns the handle to the groupSelector defined on the given componentGeneratorSelector element.

- Returns: groupSelectorID of type **String** - Handle to a groupSelector element
- Input: componentGeneratorSelectorID of type **String** - Handle to a componentGeneratorSelector element

F.7.51.2 getGeneratorChainChainGroupIDs

Description: Returns the handles to all the groups defined on the given generatorChain element.

- Returns: groupIDs of type **String** - List of handles to the group elements

- Input: generatorChainID of type **String** - Handle to a generatorChain element

F.7.51.3 getGeneratorChainChoiceIDs

Description: Returns the handles to all the choices defined on the given generatorChain object.

- Returns: choiceIDs of type **String** - List of handles to choice element
- Input: generatorChainID of type **String** - Handle to a generatorChain element

F.7.51.4 getGeneratorChainComponentGeneratorSelectorIDs

Description: Returns the handles to all the componentGeneratorSelectors defined on the given generatorChain element.

- Returns: componentGeneratorSelectorIDs of type **String** - List of handles from the componentGeneratorSelector elements
- Input: generatorChainID of type **String** - Handle to a generatorChain element

F.7.51.5 getGeneratorChainGeneratorChainSelectorIDs

Description: Returns the handles to all the generatorChainSelectors defined on the given generatorChain element.

- Returns: generatorChainSelectorIDs of type **String** - List of handles to the generatorChainSelector elements
- Input: generatorChainID of type **String** - Handle to a generatorChain element

F.7.51.6 getGeneratorChainGeneratorIDs

Description: Returns the handles to all the generators defined on the given generatorChain object.

- Returns: generatorIDs of type **String** - List of handles to generator elements
- Input: generatorChainID of type **String** - Handle to a generatorChain object

F.7.51.7 getGeneratorChainSelectorGeneratorChainRefByID

Description: Returns the handle to the generatorChain instance referenced from the given generatorChainSelector element.

- Returns: generatorChainID of type **String** - Handle to the referenced generatorChain object
- Input: generatorChainSelectorID of type **String** - Handle to a generatorChainSelector element

F.7.51.8 getGeneratorChainSelectorGeneratorChainRefByVLNV

Description: Returns the VLNV of the generatorChain referenced from the given generatorChainSelector element.

- Returns: VLNV of type **String** - The VLNV of the referenced generatorChain object
- Input: generatorChainSelectorID of type **String** - Handle to a generatorChainSelector element

F.7.51.9 getGeneratorChainSelectorGroupSelectorID

Description: Returns the handle to the groupSelector defined on the given generatorChainSelector element.

- Returns: groupSelectorID of type **String** - Handle to the groupSelector element

- Input: generatorChainSelectorID of type **String** - Handle to a generatorChainSelector element

F.7.51.10 getGroupSelectorNameIDs

Description: Returns the handles to all names defined on the given groupSelector element.

- Returns: nameIDs of type **String** - List of handles to the name elements
- Input: groupSelectorID of type **String** - Handle to a groupSelector element

F.7.51.11 getGroupSelectorSelectionNames

Description: Returns names for a given groupSelector element.

- Returns: names of type **String** - List of generator(chain) group names
- Input: groupSelectorID of type **String** - Handle to a groupSelector element

F.7.51.12 getGroupSelectorSelectionOperator

Description: Returns multipleGroupSelectionOperation for a given groupSelector element.

- Returns: operator of type **String** - Operator expression
- Input: groupSelectorID of type **String** - Handle to a groupSelector element

F.7.52 Generator chain (EXTENDED)

F.7.52.1 addGeneratorChainChainGroup

Description: Adds chainGroup with the given value to the given generatorChain element.

- Returns: status of type **String** - Indicates call is successful (true) or not (false)
- Input: generatorChainID of type **String** - Handle to a generatorChain element
- Input: group of type **String** - ChainGroup value

F.7.52.2 addGeneratorChainChoice

Description: Adds choice with the given name and enumerations to the given generatorChain element.

- Returns: choiceID of type **String** - Handle to a choice element
- Input: generatorChainID of type **String** - Handle to a generatorChain element
- Input: name of type **String** - Choice name
- Input: enumerations of type **String[]** - Choice enumeration values

F.7.52.3 addGeneratorChainComponentGeneratorSelector

Description: Adds an componentGeneratorSelector to the given generatorChain element.

- Returns: componentGeneratorSelectorID of type **String** - Handle to a new componentGeneratorSelector element
- Input: generatorChainID of type **String** - Handle to a generatorChain element
- Input: name of type **String** - groupSelector name

F.7.52.4 addGeneratorChainGenerator

Description: Adds generator with the given generatorExe to the given generatorChain element.

- Returns: generatorID of type **String** - Handle to a generator element
- Input: generatorChainID of type **String** - Handle to a generatorChain element
- Input: name of type **String** - name of the generator
- Input: generatorExecutable of type **String** - Path to generator executable

F.7.52.5 addGeneratorChainGeneratorChainSelector

Description: Adds an generatorChainSelector to the given generatorChain element.

- Returns: generatorChainSelectorID of type **String** - Handle to a new generatorChainSelector element
- Input: generatorChainID of type **String** - Handle to a generatorChain element
- Input: name of type **String** - Name of the new generatorChainSelector

F.7.52.6 addGroupSelectorName

Description: Adds an element Name to a group selector element.

- Returns: nameID of type **String** - Handle of name element
- Input: groupSelectorID of type **String** - Handle to a groupSelectorName element
- Input: name of type **String** - Handle the name of the element

F.7.52.7 removeGeneratorChainChainGroup

Description: Removes the given chainGroup.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: chainGroupID of type **String** - Handle to a chainGroup element

F.7.52.8 removeGeneratorChainChoice

Description: Removes the given choice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: choiceID of type **String** - Handle to a choice element

F.7.52.9 removeGeneratorChainComponentGeneratorSelector

Description: Removes an componentGeneratorSelector with the given componentGeneratorSelectorID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentGeneratorSelectorID of type **String** - Handle to a componentGeneratorSelector element

F.7.52.10 removeGeneratorChainGenerator

Description: Removes a generator with the given generatorID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorID of type **String** - Handle to a generator element

F.7.52.11 removeGeneratorChainGeneratorChainSelector

Description: Removes a generatorChainSelector with the given generatorChainSelectorID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: generatorChainSelectorID of type **String** - Handle to a generatorChainSelector element

F.7.52.12 removeGroupSelectorName

Description: Removes the given group selector name from its containing element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: groupSelectorNameID of type **String** - Handle to a groupSelectorName element

F.7.52.13 setComponentGeneratorSelectorGroupSelector

Description: Sets the groupSelector with the given value for the given componentGeneratorSelector element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentGeneratorSelectorID of type **String** - Handle to an componentGeneratorSelector element
- Input: names of type **String[]** - groupSelector names

F.7.52.14 setGeneratorChainSelectorGeneratorChainRef

Description: Sets the generatorChainRef with the given value for the given generatorChainSelector element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorChainSelectorID of type **String** - Handle to an generatorChainSelector element
- Input: generatorChainVLNV of type **String[]** - generatorChain reference

F.7.52.15 setGeneratorChainSelectorGroupSelector

Description: Sets the groupSelector with the given value for the given generatorChainSelector element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: generatorChainSelectorID of type **String** - Handle to an generatorChainSelector element
- Input: names of type **String[]** - groupSelector names

F.7.53 Indirect interface (BASE)

F.7.53.1 getIndirectAddressRefAddressBlockRefByName

Description: Returns the addressBlockRef defined on the given indirectAddressRef element.

- Returns: addressBlockRef of type **String** - The referenced addressBlock
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef element

F.7.53.2 getIndirectAddressRefAddressBlockRefID

Description: Returns the handle to the addressBlockRef defined on the given indirectAddressRef element.

- Returns: addressBlockRefID of type **String** - Handle to the addressBlockRef element
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef element

F.7.53.3 getIndirectAddressRefAddressSpaceRefByName

Description: Returns the addressSpaceRef defined on the given indirectAddressRef element.

- Returns: addressSpaceRef of type ***String*** - The referenced addressSpace
- Input: indirectAddressRefID of type ***String*** - Handle to an indirectAddressRef element

F.7.53.4 getIndirectAddressRefAddressSpaceRefID

Description: Returns the handle to the addressSpaceRef defined on the given indirectAddressRef element.

- Returns: addressSpaceRefID of type ***String*** - Handle to the addressSpaceRef element
- Input: indirectAddressRefID of type ***String*** - Handle to an indirectAddressRef element

F.7.53.5 getIndirectAddressRefAlternateRegisterRefByName

Description: Returns the alternateRegisterRef defined on the given indirectAddressRef element.

- Returns: alternateRegisterRef of type ***String*** - The referenced alternateRegister
- Input: indirectAddressRefID of type ***String*** - Handle to an indirectAddressRef element

F.7.53.6 getIndirectAddressRefAlternateRegisterRefID

Description: Returns the handle to the alternateRegisterRef defined on the given indirectAddressRef element.

- Returns: alternateRegisterRefID of type ***String*** - Handle to the alternateRegisterRef element
- Input: indirectAddressRefID of type ***String*** - Handle to an indirectAddressRef element

F.7.53.7 getIndirectAddressRefBankRefByNames

Description: Returns all the bankRefs defined on the given indirectAddressRef element.

- Returns: bankRefs of type ***String*** - List of the referenced banks
- Input: indirectAddressRefID of type ***String*** - Handle to an indirectAddressRef element

F.7.53.8 getIndirectAddressRefBankRefIDs

Description: Returns the handles to all the bankRefs defined on the given indirectAddressRef element.

- Returns: bankRefIDs of type ***String*** - List of handles to the bankRef elements
- Input: indirectAddressRefID of type ***String*** - Handle to an indirectAddressRef element

F.7.53.9 getIndirectAddressRefFieldRefByName

Description: Returns the FieldRef defined on the given indirectAddressRef element.

- Returns: fieldRef of type ***String*** - The referenced field
- Input: indirectAddressRefID of type ***String*** - Handle to an indirectAddressRef element

F.7.53.10 getIndirectAddressRefFieldRefID

Description: Returns the handle to the fileRef defined on the given indirectAddressRef element.

- Returns: fieldRefID of type ***String*** - Handle to the fieldRef element
- Input: indirectAddressRefID of type ***String*** - Handle to an indirectAddressRef element

F.7.53.11 getIndirectAddressRefMemoryMapRefByName

Description: Returns the memoryMapRef defined on the given indirectAddressRef element.

- Returns: `memoryMapRef` of type ***String*** - The referenced memoryMap
- Input: `indirectAddressRefID` of type ***String*** - Handle to an indirectAddressRef element

F.7.53.12 getIndirectAddressRefMemoryMapRefID

Description: Returns the handle to the memoryMapRef defined on the given indirectAddressRef element.

- Returns: `memoryMapRefID` of type ***String*** - Handle to the referenced memoryMapRef element
- Input: `indirectAddressRefID` of type ***String*** - Handle to an indirectAddressRef element

F.7.53.13 getIndirectAddressRefMemoryRemapRefByID

Description: Returns the handle to a memoryRemap from the given indirectAddressRef element.

- Returns: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element
- Input: `indirectAddressRefID` of type ***String*** - Handle to a indirectAddressRef element

F.7.53.14 getIndirectAddressRefMemoryRemapRefByName

Description: Returns the memoryRemap value from the given indirectAddressRef element.

- Returns: `memoryRemapRef` of type ***String*** - The memoryRemapRef value
- Input: `indirectAddressRefID` of type ***String*** - Handle to a indirectAddressRef element

F.7.53.15 getIndirectAddressRefMemoryRemapRefID

Description: Returns the handle to a memoryRemapRef from the given indirectAddressRef element.

- Returns: `memoryRemapRefID` of type ***String*** - Handle to a memoryRemapRef element
- Input: `indirectAddressRefID` of type ***String*** - Handle to a indirectAddressRef element

F.7.53.16 getIndirectAddressRefRegisterFileRefByNames

Description: Returns all the registerFiles defined on the given indirectAddressRef element.

- Returns: `registerFileRef` of type ***String*** - List of the referenced registerFiles
- Input: `indirectAddressRefID` of type ***String*** - Handle to an indirectAddressRef element

F.7.53.17 getIndirectAddressRefRegisterFileRefIDs

Description: Returns the handles to all the registerFileRefs defined on the given indirectAddressRef element.

- Returns: `registerFileRefIDs` of type ***String*** - List of handles to the registerFileRef elements
- Input: `indirectAddressRefID` of type ***String*** - Handle to an indirectAddressRef element

F.7.53.18 getIndirectAddressRefRegisterRefByName

Description: Returns the RegisterRef defined on the given indirectAddressRef element.

- Returns: `registerRef` of type ***String*** - The referenced register
- Input: `indirectAddressRefID` of type ***String*** - Handle to an indirectAddressRef element

F.7.53.19 getIndirectAddressRefRegisterRefID

Description: Returns the handle to the registerRef defined on the given indirectAddressRef element.

- Returns: `registerRefID` of type ***String*** - Handle to the registerRef element
- Input: `indirectAddressRefID` of type ***String*** - Handle to an indirectAddressRef element

F.7.53.20 getIndirectDataRefAddressBlockRefByName

Description: Returns the addressBlockRef defined on the given indirectDataRef element.

- Returns: `addressBlockRef` of type ***String*** - The referenced addressBlock
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.21 getIndirectDataRefAddressBlockRefID

Description: Returns the handle to the addressBlockRef defined on the given indirectDataRef element.

- Returns: `addressBlockRefID` of type ***String*** - Handle to the addressBlockRef element
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.22 getIndirectDataRefAddressSpaceRefByName

Description: Returns the addressSpaceRef defined on the given indirectDataRef element.

- Returns: `addressSpaceRef` of type ***String*** - The referenced addressSpace
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.23 getIndirectDataRefAddressSpaceRefID

Description: Returns the handle to the addressSpaceRef defined on the given indirectDataRef element.

- Returns: `addressSpaceRefID` of type ***String*** - Handle to the addressSpaceRef element
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.24 getIndirectDataRefAlternateRegisterRefByName

Description: Returns the addressBlockRef defined on the given indirectDataRef element.

- Returns: `alternateRegisterRef` of type ***String*** - The referenced alternateRegister
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.25 getIndirectDataRefAlternateRegisterRefID

Description: Returns the handle to the alternateRegisterRef defined on the given indirectDataRef element.

- Returns: `alternateRegisterRefID` of type ***String*** - Handle to the alternateRegisterRef element
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.26 getIndirectDataRefBankRefByNames

Description: Returns the bankRef defined on the given indirectDataRef element.

- Returns: `bankRef` of type ***String*** - The referenced bank
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.27 getIndirectDataRefBankRefIDs

Description: Returns the handles to all the bankRefs defined on the given indirectDataRef element.

- Returns: bankRefIDs of type ***String*** - List of handles to the bankRef elements
- Input: indirectDataRefID of type ***String*** - Handle to an indirectDataRef element

F.7.53.28 getIndirectDataRefFieldRefByName

Description: Returns the fieldRef defined on the given indirectDataRef element.

- Returns: fieldRef of type ***String*** - The referenced field
- Input: indirectDataRefID of type ***String*** - Handle to an indirectDataRef element

F.7.53.29 getIndirectDataRefFieldRefID

Description: Returns the handle to the fieldRef defined on the given indirectDataRef element.

- Returns: fieldRefID of type ***String*** - Handle to the fieldRef element
- Input: indirectDataRefID of type ***String*** - Handle to an indirectDataRef element

F.7.53.30 getIndirectDataRefMemoryMapRefByName

Description: Returns the memoryMapRef defined on the given indirectDataRef element.

- Returns: memoryMapRef of type ***String*** - The referenced memoryMap
- Input: indirectDataRefID of type ***String*** - Handle to an indirectDataRef element

F.7.53.31 getIndirectDataRefMemoryMapRefID

Description: Returns the handle to the memoryMapRef defined on the given indirectDataRef element.

- Returns: memoryMapRefID of type ***String*** - Handle to the memoryMapRef element
- Input: indirectDataRefID of type ***String*** - Handle to an indirectDataRef element

F.7.53.32 getIndirectDataRefMemoryRemapRefById

Description: Returns the handle to a memoryRemap element from the given indirectDataRef element.

- Returns: memoryRemapID of type ***String*** - Handle to a memoryRemap element
- Input: indirectDataRefID of type ***String*** - Handle to a indirectDataRef element

F.7.53.33 getIndirectDataRefMemoryRemapRefByName

Description: Returns the memoryRemapRef value from the given indirectDataRef element.

- Returns: memoryRemapRef of type ***String*** - The memoryRemapRef value
- Input: indirectDataRefID of type ***String*** - Handle to a indirectDataRef element

F.7.53.34 getIndirectDataRefMemoryRemapRefID

Description: Returns the handle to a memoryRemapRef element from the given indirectDataRef element.

- Returns: memoryRemapRefID of type ***String*** - Handle to a memoryRemapRef element
- Input: indirectDataRefID of type ***String*** - Handle to a indirectDataRef element

F.7.53.35 getIndirectDataRefRegisterFileRefByNames

Description: Returns the registerFileRef defined on the given indirectDataRef element.

- Returns: `registerFileRef` of type ***String*** - List of registerFileRefs
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.36 getIndirectDataRefRegisterFileRefIDs

Description: Returns the handles to all the registerFileRefs defined on the given indirectDataRef element.

- Returns: `registerFileRefIDs` of type ***String*** - List of handles to the registerFileRef elements
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.37 getIndirectDataRefRegisterRefByName

Description: Returns the registerRef defined on the given indirectDataRef element.

- Returns: `registerRef` of type ***String*** - The referenced register
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.38 getIndirectDataRefRegisterRefID

Description: Returns the handle to the registerRef defined on the given indirectDataRef element.

- Returns: `registerRefID` of type ***String*** - Handle to the registerRef element
- Input: `indirectDataRefID` of type ***String*** - Handle to an indirectDataRef element

F.7.53.39 getIndirectInterfaceBitsInLau

Description: Returns the bitsInLau resolved value on an indirectInterface element

- Returns: `value` of type ***Long*** - The bitsInLau resolved value
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.53.40 getIndirectInterfaceBitsInLauExpression

Description: Returns the bitsInLau expression defined on the given indirectInterface element

- Returns: `bitsInLauExpression` of type ***String*** - The bitsInLau expression
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.53.41 getIndirectInterfaceEndianness

Description: Returns the endianness defined on the given indirectInterface element.

- Returns: `value` of type ***String*** - The endianness value
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.53.42 getIndirectInterfaceIndirectAddressRefID

Description: Returns the handle to indirectAddressRef defined on the given indirectInterface element.

- Returns: `indirectAddressRefID` of type ***String*** - Handle to the indirectAddressRef element
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.53.43 getIndirectInterfaceIndirectDataRefID

Description: Returns the handle to the indirectDataRef defined on the given indirectInterface element.

- Returns: `indirectDataRefID` of type ***String*** - Handle to the indirectDataRef element
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.53.44 getIndirectInterfaceMemoryMapRefByID

Description: Returns the handle to the memoryMap referenced from the given indirectInterface element.

- Returns: `memoryMapID` of type ***String*** - Handle to the referenced memoryMap element
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.53.45 getIndirectInterfaceMemoryMapRefByName

Description: Returns the memoryMapRef defined on the given indirectInterface element.

- Returns: `memoryMapRef` of type ***String*** - The referenced memoryMap
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.53.46 getIndirectInterfaceTransparentBridgeIDs

Description: Returns the the list of handles to transparent bridges defined on the given indirectInterface element.

- Returns: `transparentBridgeIDs` of type ***String*** - List of handles to the transparentBridge elements
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.54 Indirect interface (EXTENDED)

F.7.54.1 addIndirectAddressRefBankRef

Description: Adds a bankRef element on an indirectAddressRef element.

- Returns: `bankRefID` of type ***String*** - the identifier on a bankRef element
- Input: `indirectAddressRefID` of type ***String*** - Handle to an indirectAddressRef on an indirectInterface element
- Input: `bankRef` of type ***String*** - the bankRef to add

F.7.54.2 addIndirectAddressRefRegisterFileRef

Description: Adds a registerFileRef on an indirectAddressRef element.

- Returns: `registerFileRefID` of type ***String*** - the added registerFileRef identifier
- Input: `indirectAddressRefID` of type ***String*** - Handle to an indirectAddressRef on an indirectInterface element
- Input: `registerFileRef` of type ***String*** - the registerFile reference to set on an indirectAddressRef

F.7.54.3 addIndirectDataRefBankRef

Description: Adds a bankRef element on an indirectDataRef element.

- Returns: `bankRefID` of type ***String*** - the identifier on a bankRef element

- Input: `indirectDataRefID` of type ***String*** - Handle to an `indirectAddressRef` on an `indirectDataRef` element
- Input: `bankRef` of type ***String*** - The `bankRef` to add

F.7.54.4 addIndirectDataRefRegisterFileRef

Description: Adds a `registerFileRef` on an `indirectDataRef` element.

- Returns: `registerFileRefID` of type ***String*** - The added `registerFileRef` identifier
- Input: `indirectDataRefID` of type ***String*** - Handle to an `indirectDataRef` on an `indirectInterface` element
- Input: `registerFileRef` of type ***String*** - The `registerFile` reference to set on an `indirectDataRef`

F.7.54.5 addIndirectInterfaceTransparentBridge

Description: Adds bridge with the given `busInterfaceRef` to the given `indirectInterface` element.

- Returns: `transparentBridgeID` of type ***String*** - Handle of a `transparentBridge` element
- Input: `indirectInterfaceID` of type ***String*** - Handle to an `indirectInterface` element
- Input: `initiatorRef` of type ***String*** - Value for `initiatorRef` attribute

F.7.54.6 removeAliasOfMemoryRemapRef

Description: Removes the `memoryRemap` reference on an `aliasOf` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `aliasOfID` of type ***String*** - Handle to an `aliasOf` element

F.7.54.7 removeBroadcastToAddressSpaceRef

Description: Removes the `addressSpaceRef` on an `broadcastTo` element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `broadcastToID` of type ***String*** - Handle to a `broadcastTo` element

F.7.54.8 removeIndirectAddressRefAddressBlockRef

Description: Removes `addressBlockRef` on an `indirectAddressRef` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indirectAddressRefID` of type ***String*** - Handle to an `indirectAddressRef` on an `indirectInterface` element

F.7.54.9 removeIndirectAddressRefAddressSpaceRef

Description: Removes the `addressSpaceRef` on an `indirectAddressRef` element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indirectAddressRefID` of type ***String*** - Handle to an `indirectAddressRef` element

F.7.54.10 removeIndirectAddressRefAlternateRegisterRef

Description: Removes an `alternateRegisterRef` on an `indirectAddressRef` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indirectAddressRefID` of type ***String*** - Handle of an `indirectAddressRef` element

F.7.54.11 removeIndirectAddressRefBankRef

Description: Removes a bank reference from an indirectAddressRef element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankRefID of type **String** - Handle to a bankRef element

F.7.54.12 removeIndirectAddressRefMemoryMapRef

Description: Removes the memoryMap reference on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef on an indirectInterface element

F.7.54.13 removeIndirectAddressRefMemoryRemapRef

Description: Removes the memoryRemap reference on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef on an indirectInterface element

F.7.54.14 removeIndirectAddressRefRegisterFileRef

Description: Removes the given registerFileRef from its containing indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileRefID of type **String** - Handle to a registerFileRef on an indirectAddressRef element

F.7.54.15 removeIndirectAddressRefRegisterRef

Description: Removes the registerRef on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef on an indirectInterface element

F.7.54.16 removeIndirectDataRefAddressBlockRef

Description: Removes addressBlockRef on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element

F.7.54.17 removeIndirectDataRefAddressSpaceRef

Description: Removes the addressSpace reference on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element

F.7.54.18 removeIndirectDataRefAlternateRegisterRef

Description: Removes an alternateRegisterRef on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle of an indirectDataRef element

F.7.54.19 removeIndirectDataRefBankRef

Description: Removes a bank reference from an indirectDataRef element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankRefID of type **String** - Handle to a bankRef element

F.7.54.20 removeIndirectDataRefMemoryMapRef

Description: Removes the memoryMap reference on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element

F.7.54.21 removeIndirectDataRefMemoryRemapRef

Description: Removes the memoryRemap reference on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element

F.7.54.22 removeIndirectDataRefRegisterFileRef

Description: Removes the given registerFileRef from its containing indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileRefID of type **String** - Handle to a registerFileRef on an indirectDataRef element

F.7.54.23 removeIndirectDataRefRegisterRef

Description: Removes the registerRef on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element

F.7.54.24 removeIndirectInterfaceEndianness

Description: Removes the endianness from the given indirectInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectInterfaceID of type **String** - Handle to an indirectInterface element

F.7.54.25 removeIndirectInterfaceTransparentBridge

Description: Removes the given bridge element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: transparentBridgeID of type **String** - Handle to a transparentBridge element

F.7.54.26 setIndirectAddressRefAddressBlockRef

Description: Sets addressBlockRef on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef on an indirectInterface element
- Input: addressBlockRef of type **String** - The addressBlock reference to set

F.7.54.27 setIndirectAddressRefAddressSpaceRef

Description: Sets the addressSpace reference on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef element
- Input: addressSpaceRef of type **String** - Name of the referenced addressSpace on an indirectAddressRef element

F.7.54.28 setIndirectAddressRefAlternateRegisterRef

Description: Sets an alternateRegisterRef on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle of an indirectAddressRef element
- Input: alternateRegisterRef of type **String** - The alternateRegisterRef value to set

F.7.54.29 setIndirectAddressRefFieldRef

Description: Sets the fieldRef reference on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef element
- Input: fieldRef of type **String** - Name of the referenced field on an indirectAddressRef element

F.7.54.30 setIndirectAddressRefMemoryMapRef

Description: Sets the memoryMap reference on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef element
- Input: memoryMapRef of type **String** - Name of the referenced memoryMap on an indirectAddressRef element

F.7.54.31 setIndirectAddressRefMemoryRemapRef

Description: Sets memoryRemapRef on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef on an indirectInterface element
- Input: memoryRemapRef of type **String** - the memoryRemap reference to set

F.7.54.32 setIndirectAddressRefRegisterRef

Description: Sets the registerRef on an indirectAddressRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectAddressRefID of type **String** - Handle to an indirectAddressRef on an indirectInterface element
- Input: registerRef of type **String** - Name of the referenced register element

F.7.54.33 setIndirectDataRefAddressBlockRef

Description: Sets addressBlockRef on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element
- Input: addressBlockRef of type **String** - the addressBlock reference to set

F.7.54.34 setIndirectDataRefAddressSpaceRef

Description: Sets the addressSpace reference on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element
- Input: addressSpaceRef of type **String** - Name of the referenced addressSpace on an indirectDataRef element

F.7.54.35 setIndirectDataRefAlternateRegisterRef

Description: Sets an alternateRegisterRef on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle of an indirectDataRef element
- Input: alternateRegisterRef of type **String** - the alternateRegisterRef value to set

F.7.54.36 setIndirectDataRefFieldRef

Description: Sets the fieldRef reference on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef element
- Input: fieldRef of type **String** - Name of the referenced field on an indirectDataRef element

F.7.54.37 setIndirectDataRefMemoryMapRef

Description: Sets the memoryMap reference on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element
- Input: memoryMapRef of type **String** - Name of the referenced memoryMap on an indirectDataRef element

F.7.54.38 setIndirectDataRefMemoryRemapRef

Description: Sets memoryRemapRef on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element
- Input: memoryRemapRef of type **String** - The memoryRemap reference to set

F.7.54.39 setIndirectDataRefRegisterRef

Description: Sets the registerRef on an indirectDataRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectDataRefID of type **String** - Handle to an indirectDataRef on an indirectInterface element
- Input: registerRef of type **String** - Name of the referenced register

F.7.54.40 setIndirectInterfaceEndianness

Description: Sets the endianness with the given value for the given indirectInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectInterfaceID of type **String** - Handle to an indirectInterface element
- Input: value of type **String** - Endianness value

F.7.54.41 setIndirectInterfaceMemoryMapRef

Description: Sets the memoryMapRef on an indirectInterface element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indirectInterfaceID of type **String** - Handle to an indirectInterface element
- Input: memoryMapRef of type **String** - The memoryMap reference to be set

F.7.55 Instantiation (BASE)

F.7.55.1 getAbstractionTypeAbstractionRefID

Description: Returns the handle to the abstractionRef defined on the given abstractionType element.

- Returns: abstractionRefID of type **String** - Handle to the abstractionRef element
- Input: abstractionTypeID of type **String** - Handle to an abstractionType element

F.7.55.2 getAbstractorInstanceAbstractorRefID

Description: Returns the handle to the abstractorRef defined on the given abstractorInstance element.

- Returns: abstractorRefID of type **String** - Handle to the abstractorRef element
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element

F.7.55.3 getComponentInstanceComponentRefID

Description: Returns the handle to the componentRef defined on the given componentInstance element.

- Returns: componentRefID of type **String** - Handle to the componentRef element
- Input: componentInstanceID of type **String** - Handle to a componentInstance element

F.7.55.4 getComponentInstantiationArchitectureName

Description: Returns the architectureName defined on the given componentInstantiation element.

- Returns: `architectureName` of type ***String*** - The architecture name
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.55.5 getComponentInstantiationClearboxElementRefIDs

Description: Returns the handles to all the clearboxElementRefs defined on the given componentInstantiation element.

- Returns: `clearboxElementRefIDs` of type ***String*** - List of handles to the clearboxElementRef elements
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.55.6 getComponentInstantiationConfigurationName

Description: Returns the configurationName defined on the given componentInstantiation element.

- Returns: `configurationName` of type ***String*** - The configuration name
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.55.7 getComponentInstantiationConstraintSetRefIDs

Description: Returns the handles to all the constraintSetRefs defined on the given componentInstantiation element.

- Returns: `constraintSetRefIDs` of type ***String*** - List of handles to constraintSetRef elements
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.55.8 getComponentInstantiationDefaultFileBuilderIDs

Description: Returns the handles to all the defaultFileBuilders defined on the given componentInstantiation element.

- Returns: `fileBuilderIDs` of type ***String*** - List of handles to fileBuilder elements
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.55.9 getComponentInstantiationFileSetRefIDs

Description: Returns the handles to all the fileSetRefs defined on the given componentInstantiation element.

- Returns: `fileSetRefIDs` of type ***String*** - List of handles to fileSetRef elements
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.55.10 getComponentInstantiationIsVirtual

Description: Returns the isVirtual element defined on the given componentInstantiation element.

- Returns: `value` of type ***Boolean*** - The isVirtual value

- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.55.11 getComponentInstantiationLanguage

Description: Returns the language defined on the given componentInstantiation element.

- Returns: language of type **String** - The language defined
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.55.12 getComponentInstantiationLanguageID

Description: Returns the handle to the language defined on the given componentInstantiation element.

- Returns: language of type **String** - Handle to the language element
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.55.13 getComponentInstantiationLibraryName

Description: Returns the libraryName defined on the given componentInstantiation element.

- Returns: libraryName of type **String** - The library name
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.55.14 getComponentInstantiationModuleName

Description: Returns the moduleName defined on the given componentInstantiation element.

- Returns: moduleName of type **String** - The module name
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.55.15 getComponentInstantiationPackageName

Description: Returns the packageName defined on the given componentInstantiation element.

- Returns: packageName of type **String** - The package name
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.55.16 getDesignConfigurationInstantiationDesignConfigurationRefByID

Description: Returns the handle to the designConfiguration instance referenced from the given designConfigurationInstantiation element.

- Returns: designConfigurationID of type **String** - Handle to the referenced designConfiguration object
- Input: designConfigurationInstantiationID of type **String** - Handle to a designConfigurationInstantiation element

F.7.55.17 getDesignConfigurationInstantiationDesignConfigurationRefByVLNV

Description: Returns the VLNV of the designConfiguration referenced from the given designConfigurationInstantiation element.

- Returns: VLVN of type **String** - The VLVN of the referenced designConfiguration object
- Input: designConfigurationInstantiationID of type **String** - Handle to a designConfigurationInstantiation element

F.7.55.18 getDesignConfigurationInstantiationDesignConfigurationRefID

Description: Returns the handle to the designConfigurationRef defined on the given designConfigurationInstantiation element.

- Returns: designConfigurationRefID of type **String** - Handle to the designConfigurationRef element
- Input: designConfigurationInstantiationID of type **String** - Handle to a designConfigurationInstantiation element

F.7.55.19 getDesignConfigurationInstantiationLanguage

Description: Returns the language defined on the given designConfigurationInstantiation element.

- Returns: language of type **String** - The language defined
- Input: designConfigurationInstantiationID of type **String** - Handle to a designConfigurationInstantiation element

F.7.55.20 getDesignConfigurationInstantiationLanguageID

Description: Returns the handle to the language defined on the given designConfigurationInstantiation element.

- Returns: language of type **String** - Handle to the language element
- Input: designConfigurationInstantiationID of type **String** - Handle to a designConfigurationInstantiation element

F.7.55.21 getDesignInstantiationDesignRefByID

Description: Returns the handle to the design instance referenced from the given designInstantiation element.

- Returns: designID of type **String** - Handle to the referenced design object
- Input: designInstantiationID of type **String** - Handle to a designInstantiation element

F.7.55.22 getDesignInstantiationDesignRefByVLNV

Description: Returns the VLVN of the design referenced from the given designInstantiation element.

- Returns: VLVN of type **String** - The VLVN of the referenced design object
- Input: designInstantiationID of type **String** - Handle to a designInstantiation element

F.7.55.23 getDesignInstantiationDesignRefID

Description: Returns the handle to the designRef defined on the given designInstantiation element.

- Returns: designRefID of type **String** - Handle to the designRef element
- Input: designInstantiationID of type **String** - Handle to a designInstantiation element

F.7.55.24 getExternalTypeDefinitionsTypeDefinitionsRefID

Description: Returns the handle to the typeDefinitionsRef defined on the given externalTypeDefinitions element.

- Returns: typeDefinitionsRefID of type **String** - Handle to the typeDefinitionsRef element
- Input: externalTypeDefinitionsID of type **String** - Handle to an externalTypeDefinitions element

F.7.55.25 getFileSetRefLocalName

Description: Returns the localName defined on the given fileSetRef element.

- Returns: localName of type **String** - The local name
- Input: fileSetRefID of type **String** - Handle to a fileSetRef element

F.7.55.26 getGeneratorChainSelectorGeneratorChainRefID

Description: Returns the handle to the generatorChainRef defined on the given generatorChainSelector element.

- Returns: generatorChainRefID of type **String** - Handle to the generatorChainRef element
- Input: generatorChainSelectorID of type **String** - Handle to a generatorChainSelector element

F.7.56 Instantiation (EXTENDED)

F.7.56.1 addComponentInstantiationClearboxElementRef

Description: Adds clearboxElementRef with the given name, pathSegmentName, and indices to the given componentInstantiation element

- Returns: clearboxElementRefID of type **String** - Handle to a new clearboxElementRef
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: name of type **String** - The clearboxElementRef name
- Input: pathSegmentValue of type **String** - The clearboxElementRef pathSegment value

F.7.56.2 addComponentInstantiationConstraintSetRef

Description: Adds constraintRefSet with the given localName to the given componentInstantiation element.

- Returns: constraintSetRefID of type **String** - Handle to a new constraintSetRef
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: localName of type **String** - ConstraintSetRef localName

F.7.56.3 addComponentInstantiationDefaultFileBuilder

Description: Adds defaultFileBuilder with the given fileType to the given componentInstantiation element.

- Returns: fileBuilderID of type **String** - Handle to a new fileBuilder
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: fileType of type **String** - The defaultFileBuilder fileType

F.7.56.4 addComponentInstantiationFileSetRef

Description: Adds fileSetRef with the given localName to the given componentInstantiation element.

- Returns: fileSetRefID of type **String** - Handle to a new fileSetRef
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: localName of type **String** - The fileSetRef localName

F.7.56.5 removeComponentInstantiationArchitectureName

Description: Removes architectureName from the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.56.6 removeComponentInstantiationClearboxElementRef

Description: Removes the given clearboxElementRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: clearboxElementRefID of type **String** - Handle to a clearboxElementRef element

F.7.56.7 removeComponentInstantiationConfigurationName

Description: Removes configurationName from the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.56.8 removeComponentInstantiationConstraintSetRef

Description: Removes the given constraintSetRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetRefID of type **String** - Handle to a constraintSetRef element

F.7.56.9 removeComponentInstantiationDefaultFileBuilder

Description: Removes the given defaultFileBuilder element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileBuilderID of type **String** - Handle to a fileBuilder element

F.7.56.10 removeComponentInstantiationFileSetRef

Description: Removes the given fileSetRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fileSetRefID of type **String** - Handle to a fileSetRef element

F.7.56.11 removeComponentInstantiationIsVirtual

Description: Removes isVirtual from the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.56.12 removeComponentInstantiationLanguage

Description: Removes language from the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.56.13 removeComponentInstantiationLibraryName

Description: Removes libraryName from the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.56.14 removeComponentInstantiationModuleName

Description: Removes moduleName from the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.56.15 removeComponentInstantiationPackageName

Description: Removes packageName from the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element

F.7.56.16 removeDesignConfigurationInstantiationLanguage

Description: Removes language from the given designConfigurationInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: designConfigurationInstantiationID of type **String** - Handle to a designConfigurationInstantiation element

F.7.56.17 setComponentInstantiationArchitectureName

Description: Sets architectureName with the given value for the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: architectureName of type **String** - ArchitectureName value

F.7.56.18 setComponentInstantiationConfigurationName

Description: Sets configurationName with the given value for the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: configurationName of type **String** - ConfigurationName value

F.7.56.19 setComponentInstantiationIsVirtual

Description: Sets isVirtual with the given value for the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: value of type **Boolean** - isVirtual value

F.7.56.20 setComponentInstantiationLanguage

Description: Sets language with the given value for the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: language of type **String** - Language value

F.7.56.21 setComponentInstantiationLibraryName

Description: Sets libraryName with the given value for the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: libraryName of type **String** - LibraryName value

F.7.56.22 setComponentInstantiationModuleName

Description: Sets moduleName with the given value for the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: moduleName of type **String** - ModuleName value

F.7.56.23 setComponentInstantiationPackageName

Description: Sets packageName with the given value for the given componentInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: componentInstantiationID of type **String** - Handle to a componentInstantiation element
- Input: packageName of type **String** - PackageName value

F.7.56.24 setDesignConfigurationInstantiationDesignConfigurationRef

Description: Sets the designConfigurationRef with the given value for the given designConfigurationInstantiation element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: `designConfigurationInstantiationID` of type ***String*** - Handle to an `designConfigurationInstantiation` element
- Input: `designConfigurationVLNV` of type ***String[]*** - The `designConfiguration` reference

F.7.56.25 setDesignConfigurationInstantiationLanguage

Description: Sets language with the given value for the given `designConfigurationInstantiation` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `designConfigurationInstantiationID` of type ***String*** - Handle to a `designConfigurationInstantiation` element
- Input: `language` of type ***String*** - The `designConfigurationInstantiation` language

F.7.56.26 setDesignInstantiationDesignRef

Description: Sets the `designRef` with the given value for the given `designInstantiation` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `designInstantiationID` of type ***String*** - Handle to an `designInstantiation` element
- Input: `designVLNV` of type ***String[]*** - The design reference

F.7.57 Memory map (BASE)

F.7.57.1 getAddressBlockAccessPolicyIDs

Description: Returns the handles to all the `accessPolicies` defined on the given `addressBlock` element.

- Returns: `accessPoliciesIDs` of type ***String*** - List of handles to the `accessPolicies` elements
- Input: `addressBlockID` of type ***String*** - Handle to an `addressBlock` element

F.7.57.2 getAddressBlockAddressBlockDefinitionRefByExternalTypeDefID

Description: Returns the handle to the `externalTypeDefinitions` referenced by the `typeDefinitions` attribute of the `addressBlockDefinitionRef` defined on the given `addressBlock` element.

- Returns: `externalTypeDefinitionsID` of type ***String*** - Handle to the `externalTypeDefinitions` element referenced by the `typeDefinitions` attribute
- Input: `addressBlockID` of type ***String*** - Handle to an `addressBlock` element

F.7.57.3 getAddressBlockAddressBlockDefinitionRefByID

Description: Returns the handle to the `addressBlockDefinition` (defined in the `typeDefinitions` root object) referenced from the given `addressBlock` element.

- Returns: `addressBlockDefinitionID` of type ***String*** - Handle to the referenced `addressBlockDefinition`
- Input: `addressBlockID` of type ***String*** - Handle to an `addressBlock` element

F.7.57.4 getAddressBlockAddressBlockDefinitionRefByName

Description: Returns the `addressBlockDefinition` referenced from the given `addressBlock` element.

- Returns: `addressBlockDefinitionRef` of type ***String*** - The referenced `addressBlockDefinition`
- Input: `addressBlockID` of type ***String*** - Handle to an `addressBlock` element

F.7.57.5 getAddressBlockAddressBlockDefinitionRefID

Description: Returns the handle to the addressBlockDefinitionRef element defined in the given addressBlock element.

- Returns: addressBlockDefinitionRefID of type **String** - Handle to the addressBlockDefinitionRef element
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.57.6 getAddressBlockArrayID

Description: Returns the handle to the array defined on the given addressBlock element.

- Returns: arrayID of type **String** - Handle to the addressBlock array
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.57.7 getAddressBlockBaseAddress

Description: Returns the baseAddress defined on the given addressBlock element.

- Returns: baseAddress of type **Long** - The base address
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.57.8 getAddressBlockBaseAddressExpression

Description: Returns the baseAddress expression defined on the given addressBlock element.

- Returns: baseAddressExpression of type **String** - The baseAddress expression
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.57.9 getAddressBlockBaseAddressID

Description: Returns the handle to the baseAddress defined on the given addressBlock element.

- Returns: baseAddressID of type **String** - Handle to the baseAddress element
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.57.10 getAddressBlockRange

Description: Returns the range value defined on the given address Block element.

- Returns: range of type **Long** - The addressBlock range value
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.57.11 getAddressBlockRangeExpression

Description: Returns the range expression defined on the given addressBlock element.

- Returns: rangeExpression of type **String** - The range expression
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.57.12 getAddressBlockRangeID

Description: Returns the handle to the range defined on the given addressBlock element.

- Returns: rangeID of type **String** - Handle to the range element
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.57.13 getAddressBlockRefIndexIDs

Description: Returns the handles to all the indices defined on the given addressBlockRef element.

- Returns: `indexID` of type ***String*** - List of handles to the index elements
- Input: `addressBlockRefID` of type ***String*** - Handle to an addressBlockRef element

F.7.57.14 getAddressBlockRegisterFileIDs

Description: Returns the handles to all the registerFiles defined on the given addressBlock element.

- Returns: `registerFileIDs` of type ***String*** - List of handles to registerFile elements
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.57.15 getAddressBlockRegisterIDs

Description: Returns the handles to all the registers defined on the given addressBlock element.

- Returns: `registerIDs` of type ***String*** - List of handles to register elements
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.57.16 getAddressBlockTypeIdentifier

Description: Returns the typeIdentifier defined on the given addressBlock element.

- Returns: `typeIdentifier` of type ***String*** - The typeIdentifier value
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.57.17 getAddressBlockUsage

Description: Returns the usage defined on the given addressBlock element.

- Returns: `usage` of type ***String*** - The addressBlock usage
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.57.18 getAddressBlockVolatility

Description: Returns the volatile value defined on the given addressBlock element.

- Returns: `volatile` of type ***Boolean*** - The volatile value
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.57.19 getAddressBlockWidth

Description: Returns the width value defined on the given addressBlock element.

- Returns: `width` of type ***Long*** - The addressBlock width value
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.57.20 getAddressBlockWidthExpression

Description: Returns the width expression defined on the given addressBlock element.

- Returns: `widthExpression` of type ***String*** - The width expression
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.57.21 getAddressBlockWidthID

Description: Returns the handle to the width defined on the given addressBlock element.

- Returns: widthID of type ***String*** - Handle to the width element
- Input: addressBlockID of type ***String*** - Handle to an addressBlock element

F.7.57.22 getAliasOfAddressBlockRefByName

Description: Returns the addressBlockRef defined on the given aliasOf element.

- Returns: addressBlockRef of type ***String*** - The referenced addressBlock
- Input: aliasOfID of type ***String*** - Handle to an aliasOf element

F.7.57.23 getAliasOfAddressBlockRefID

Description: Returns the handle to the addressBlockRef defined on the given aliasOf element.

- Returns: addressBlockRefID of type ***String*** - Handle to the addressBlockRef element
- Input: aliasOfID of type ***String*** - Handle to an aliasOf element

F.7.57.24 getAliasOfBankRefByNames

Description: Returns all the bankRefs defined on the given aliasOf element.

- Returns: bankRef of type ***String*** - List of bankRef names
- Input: aliasOfID of type ***String*** - Handle to an aliasOf element

F.7.57.25 getAliasOfBankRefIDs

Description: Returns the handles to the bankRefs defined on the given aliasOf element.

- Returns: bankRefID of type ***String*** - List of handles to the bankRef elements
- Input: aliasOfID of type ***String*** - Handle to an aliasOf element

F.7.57.26 getAliasOfMemoryMapRefByName

Description: Returns the memoryMapRef defined on the given aliasOf element.

- Returns: memoryMapRef of type ***String*** - The referenced memoryMap
- Input: aliasOfID of type ***String*** - Handle to an aliasOf element

F.7.57.27 getAliasOfMemoryMapRefID

Description: Returns the handle to the memoryMapRef defined on the given AliasOf element.

- Returns: memoryMapRefID of type ***String*** - Handle to the memoryMapRef element
- Input: aliasOfID of type ***String*** - Handle to an aliasOf element

F.7.57.28 getAliasOfMemoryRemapRefByID

Description: Returns a handle to a memoryRemap defined on the given aliasOf element.

- Returns: memoryMapID of type ***String*** - Handle to a memoryRemap
- Input: aliasOfID of type ***String*** - Handle to an aliasOf element

F.7.57.29 getAliasOfMemoryRemapRefByName

Description: Returns the memoryRemapRef defined on the given aliasOf element.

- Returns: memoryRemapRef of type **String** - The memoryRemap reference
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.57.30 getAliasOfMemoryRemapRefID

Description: Returns a handle to a memoryRemapRef defined on the given aliasOf element.

- Returns: memoryRemapRefID of type **String** - Handle to a memoryRemapRef
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.57.31 getBankAddressBlockIDs

Description: Returns the handles to all the addressBlocks defined on the given bank element.

- Returns: addressBlockIDs of type **String** - List of handles to the addressBlock elements
- Input: bankID of type **String** - Handle to a bank element

F.7.57.32 getBankBankDefinitionRefByExternalTypeDefID

Description: Returns the handle to the externalTypeDefinitions referenced by the typeDefinitions attribute of the bankDefinitionRef defined on the given bank element.

- Returns: bankDefinitionID of type **String** - Handle to the externalTypeDefinitions element referenced by the typeDefinitions attribute
- Input: bankID of type **String** - Handle to a bank element

F.7.57.33 getBankBankDefinitionRefByID

Description: Returns the handle to the bankDefinition referenced from the given bank element.

- Returns: bankDefinitionID of type **String** - Handle to the referenced bankDefinition element
- Input: bankID of type **String** - Handle to a bank element

F.7.57.34 getBankBankDefinitionRefByName

Description: Returns the bankDefinitionRef defined on the given bank element.

- Returns: bankDefinitionRef of type **String** - The referenced bankDefinition
- Input: bankID of type **String** - Handle to a bank element

F.7.57.35 getBankBankDefinitionRefID

Description: Returns the handle to the bankDefinitionRef defined on the given bank element.

- Returns: bankDefinitionRefID of type **String** - Handle to the bankDefinitionRef element
- Input: bankID of type **String** - Handle to a bank element

F.7.57.36 getBankBankIDs

Description: Returns the handles to all the banks defined on the given bank element.

- Returns: bankIDs of type **String** - List of handles to the bank elements
- Input: bankID of type **String** - Handle to a bank element

F.7.57.37 getBankBaseAddress

Description: Returns the baseAddress defined on the given bank or localBank element.

- Returns: `baseAddress` of type **Long** - The base address
- Input: `bankOrLocalBankID` of type **String** - Handle to a bank or localBank element

F.7.57.38 getBankBaseAddressExpression

Description: Returns the baseAddress expression defined on the given bank or localBank element.

- Returns: `baseAddressExpression` of type **String** - The baseAddress expression
- Input: `bankOrLocalBankID` of type **String** - Handle to a bank or localBank element

F.7.57.39 getBankBaseAddressID

Description: Returns the handle to the baseAddress defined on the given bank element.

- Returns: `baseAddressID` of type **String** - Handle to the baseAddress element
- Input: `bankID` of type **String** - Handle to a bank element

F.7.57.40 getBankBlockAndSubspaceElementIDs

Description: Returns the handles to all the banks, addressBlocks, or subspaceMaps defined on given bank or localBank element.

- Returns: `memoryElementIDs` of type **String** - List of handles to the bank, addressBlock, or subspaceMap elements
- Input: `bankOrLocalBankID` of type **String** - Handle to a bank or localBank element

F.7.57.41 getBankSubspaceMapIDs

Description: Returns the handles to all the subspaceMaps defined on the given bank element.

- Returns: `subspaceMapIDs` of type **String** - List of handles to the subspaceMap elements
- Input: `bankID` of type **String** - Handle to a bank element

F.7.57.42 getBankUsage

Description: Returns the usage defined on the given bank or localBank element.

- Returns: `usage` of type **String** - the bank usage
- Input: `bankOrLocalBankID` of type **String** - Handle to a bank or localBank element

F.7.57.43 getBankVolatility

Description: Returns the volatile value defined on the given bank or localBank element.

- Returns: `volatile` of type **Boolean** - The volatile value
- Input: `bankOrLocalBankID` of type **String** - Handle to a bank or localBank element

F.7.57.44 getMemoryMapAddressBlockIDs

Description: Returns the handles to all the addressBlocks defined on the given memoryMap element.

- Returns: `addressBlockIDs` of type **String** - List of handles to the addressBlock elements
- Input: `memoryMapID` of type **String** - Handle to a memoryMap element

F.7.57.45 getMemoryMapAddressUnitBits

Description: Returns the addressUnitBits defined on the given memoryMap or localMemoryMap element.

- Returns: addressUnitBits of type **Long** - The addressUnitBits value
- Input: memoryMapOrLocalMemoryMapID of type **String** - Handle to a memoryMap or localMemoryMap element

F.7.57.46 getMemoryMapAddressUnitBitsExpression

Description: Returns the addressUnitBits expression defined on the given memoryMap or localMemoryMap element.

- Returns: addressUnitBits of type **String** - The addressUnitBits expression
- Input: memoryMapOrLocalMemoryMapID of type **String** - Handle to a memoryMap or localMemoryMap element

F.7.57.47 getMemoryMapAddressUnitBitsID

Description: Returns the handle to the addressUnitBits defined on the given memoryMap element.

- Returns: addressUnitBitsID of type **String** - Handle to the addressUnitBits element
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.57.48 getMemoryMapBankIDs

Description: Returns the handles to all the banks defined on the given memoryMap element.

- Returns: bankIDs of type **String** - List of handles to the bank elements
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.57.49 getMemoryMapElementIDs

Description: Returns the handles to all the banks, addressBlocks, or subspaceMaps defined on given memoryMap or localMemoryMap element.

- Returns: memoryMapElementIDs of type **String** - List of handles to bank, addressBlock, or subspaceMap elements
- Input: memoryMapOrLocalMemoryMapID of type **String** - Handle to a memoryMap or localMemoryMap element

F.7.57.50 getMemoryMapElementType

Description: Returns the type of the given memoryMap element.

- Returns: memoryType of type **String** - The memory type (addressBlock, bank, or subSpaceMap)
- Input: memoryMapElementID of type **String** - Handle to a memoryMap element

F.7.57.51 getMemoryMapMemoryMapDefinitionRefByExternalTypeDefID

Description: Returns the handle to the externalTypeDefinitions referenced by the typeDefinitions attribute of the memoryMapDefinitionRef defined on the given memoryMap element.

- Returns: externalTypeDefinitionsID of type **String** - Handle to the externalTypeDefinitions element referenced by the typeDefinitions attribute
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.57.52 getMemoryMapMemoryMapDefinitionRefByID

Description: Returns the handle to the memoryMapDefinition (defined in the typeDefinitions root object) referenced from the given memoryMap element.

- Returns: memoryMapDefinitionID of type **String** - Handle to the referenced memoryMapDefinition
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.57.53 getMemoryMapMemoryMapDefinitionRefByName

Description: Returns the memoryMapDefinitionRef defined on the given memoryMap element.

- Returns: memoryMapDefinitionRef of type **String** - The referenced memoryMapDefinition
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.57.54 getMemoryMapMemoryMapDefinitionRefID

Description: Returns the handle to the memoryMapDefinitionRef defined on the given memoryMap element.

- Returns: memoryMapDefinitionRefID of type **String** - Handle to the memoryMapDefinitionRef element
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.57.55 getMemoryMapMemoryRemapIDs

Description: Returns the handles to all the memory Remaps defined on the given memoryMap element.

- Returns: memoryRemapIDs of type **String** - List of handles to the memoryRemap elements
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.57.56 getMemoryMapRemapIDs

Description: Returns the handles to all the memoryRemaps defined on the given memoryMap or localMemoryMap element.

- Returns: memoryRemapIDs of type **String** - List of handles to memoryRemap elements
- Input: memoryMapOrLocalMemoryMapID of type **String** - Handle to a memoryMap or localMemoryMap element

F.7.57.57 getMemoryMapShared

Description: Returns the shared value defined on the given memoryMap or localMemoryMap element.

- Returns: shared of type **String** - The shared value
- Input: memoryMapOrLocalMemoryMapID of type **String** - Handle to a memoryMap or a localMemoryMap element

F.7.57.58 getMemoryMapSubspaceMapIDs

Description: Returns the handles to all the subSpaceMaps defined on the given memoryMap element.

- Returns: subSpaceMapID of type **String** - List of handles to the subSpaceMap elements
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.57.59 getMemoryRemapAddressBlockIDs

Description: Returns the handles to all the addressBlocks defined on the given memoryRemap element.

- Returns: `addressBlockIDs` of type ***String*** - List of handles to the addressBlock elements
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.57.60 getMemoryRemapBankIDs

Description: Returns the handles to all the banks defined on the given memoryRemap element.

- Returns: `bankIDs` of type ***String*** - List of handles to the bank elements
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.57.61 getMemoryRemapElementIDs

Description: Returns the handles to all the banks, addressBlocks, or subspaceMaps defined on the given memoryRemap element.

- Returns: `memoryMapElementIDs` of type ***String*** - List of handles to bank, addressBlock, or subspaceMap elements
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.57.62 getMemoryRemapModeRefByID

Description: Returns the modeID defined on the given memoryRemap element.

- Returns: `modeID` of type ***String*** - Handle to the referenced mode element
- Input: `memoryRemapID` of type ***String*** - Handle to an memoryRemap element
- Input: `modeRef` of type ***String*** - Handle to the referenced modeRef element

F.7.57.63 getMemoryRemapModeRefByNames

Description: Returns all the modeRefs defined on the given memoryRemap element.

- Returns: `modeRefs` of type ***String*** - List of the referenced modes
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.57.64 getMemoryRemapModeRefIDs

Description: Returns the handles to all the modeRefs defined on the given memoryMap element.

- Returns: `modeRefIDs` of type ***String*** - List of handles to the modeRef elements
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.57.65 getMemoryRemapRemapDefinitionRefByExternalTypeDefID

Description: Returns the handle to the externalTypeDefinitions referenced by the typeDefinitions attribute of the remapDefinitionRef defined on the given memoryRemap element.

- Returns: `remapDefinitionRefID` of type ***String*** - Handle to the externalTypeDefinitions element referenced by the typeDefinitions attribute
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.57.66 getMemoryRemapRemapDefinitionRefByID

Description: Returns the handle to the remapDefinition referenced from the given memoryRemap element.

- Returns: `remapDefinitionRefID` of type ***String*** - Handle to the referenced `remapDefinition` element
- Input: `memoryRemapID` of type ***String*** - Handle to a `memoryRemap` element

F.7.57.67 getMemoryRemapRemapDefinitionRefByName

Description: Returns the `remapDefinitionRef` defined on the given `memoryRemap` element.

- Returns: `remapDefinitionRef` of type ***String*** - The `remapDefinitionRef` on a `memoryRemap` element
- Input: `memoryRemapID` of type ***String*** - Handle to a `memoryRemap` element

F.7.57.68 getMemoryRemapRemapDefinitionRefID

Description: Returns the handle to the `remapDefinitionRef` defined on the given `memoryRemap` element.

- Returns: `remapDefinitionRefID` of type ***String*** - Handle to the `remapDefinitionRef` element
- Input: `memoryRemapID` of type ***String*** - Handle to a `memoryRemap` element

F.7.57.69 getMemoryRemapSubspaceMapIDs

Description: Returns the handles to all the `subspaceMaps` defined on the given `memoryRemap` element.

- Returns: `subspaceMapID` of type ***String*** - List of handles to the `subspaceMap` elements
- Input: `memoryRemapID` of type ***String*** - Handle to a `memoryRemap` element

F.7.57.70 getSubspaceMapBaseAddress

Description: Returns the `baseAddress` (resolved) value defined on the given `subspaceMap`.

- Returns: `baseAddress` of type ***Long*** - The base address value
- Input: `subSpaceMapID` of type ***String*** - Handle to a `subspaceMap` element

F.7.57.71 getSubspaceMapBaseAddressExpression

Description: Returns the `baseAddress` expression defined on the given `subspaceMap`.

- Returns: `baseAddress` of type ***String*** - The base address expression
- Input: `subSpaceMapID` of type ***String*** - Handle to a `subspaceMap` element

F.7.57.72 getSubspaceMapBaseAddressID

Description: Returns the handle to the `baseAddress` defined on the given `subspaceMap` element.

- Returns: `baseAddressID` of type ***String*** - Handle to the `baseAddress` element
- Input: `subspaceMapID` of type ***String*** - Handle to a `subspaceMap` element

F.7.57.73 getSubspaceMapSegmentRefByName

Description: Returns the `segmentRef` defined on the given `subSpace` element.

- Returns: `segmentRef` of type ***String*** - The referenced segment
- Input: `subSpaceID` of type ***String*** - Handle to a `subSpace` element

F.7.58 Memory map (EXTENDED)

F.7.58.1 addAccessPolicyModeRef

Description: Adds a modeRef on an accessPolicy element.

- Returns: modeRefID of type **String** - Handle to the added modeRef
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element
- Input: modeRef of type **String** - Name of the referenced access mode
- Input: priority of type **Long** - The non negative integer indication the priority

F.7.58.2 addAddressBlockRefIndex

Description: Adds an index to an addressBlockRef element.

- Returns: indexID of type **String** - Handle to the added index
- Input: addressBlockRefID of type **String** - Handle to an addressBlockRef element
- Input: value of type **String** - Index value

F.7.58.3 addAddressBlockRegister

Description: Adds a register with the given name, offset, and size, and a field with the given name, offset, and width to the given addressBlock.

- Returns: registerID of type **String** - Handle to a new register element
- Input: addressBlockID of type **String** - Handle to an addressBlock element
- Input: name of type **String** - Register name
- Input: addressOffset of type **String** - Register address offset
- Input: size of type **String** - Register size
- Input: fieldName of type **String** - Field name
- Input: fieldOffset of type **String** - Field offset
- Input: fieldWidth of type **String** - Field width

F.7.58.4 addAddressBlockRegisterFile

Description: Adds a registerFile with the given name, addressOffset, and range to the given addressBlock element.

- Returns: registerFileID of type **String** - Handle to a new registerFile element
- Input: addressBlockID of type **String** - Handle to an addressBlock element
- Input: name of type **String** - RegisterFile name
- Input: addressOffset of type **String** - RegisterFile address offset
- Input: range of type **String** - RegisterFile range

F.7.58.5 addAliasOfBankRef

Description: Adds an bankRef on an aliasOf element.

- Returns: bankRefID of type **String** - Handle to a bankRef element
- Input: aliasOfID of type **String** - Handle to an aliasOf element
- Input: bankRef of type **String** - Name of the referenced bank

F.7.58.6 addBankAccessPolicy

Description: Adds an accessPolicy on a bank element.

- Returns: accessPolicyID of type **String** - the accessPolicy identifier on a bank element
- Input: bankID of type **String** - Handle to a bank element
- Input: access of type **String** - Access enumerated value. Can be one of: read-only, write-only, read-write, writeOnce, read-writeOnce or no-access

F.7.58.7 addBankAddressBlock

Description: Adds an addressBlock with the given name, range, and width to the given bank element.

- Returns: addressBlockID of type **String** - Handle to a new addressBlock element
- Input: bankID of type **String** - Handle to a bank element
- Input: name of type **String** - AddressBlock name
- Input: range of type **String** - AddressBlock range expression
- Input: width of type **String** - AddressBlock width expression

F.7.58.8 addBankBank

Description: Adds a bank to the given bank element

- Returns: bankID of type **String** - Handle to a bank identifier
- Input: bankID of type **String** - Handle to the given bank element
- Input: name of type **String** - Bank name
- Input: bankAlignment of type **String** - BankAlignment
- Input: addressBlockName of type **String** - The addressBlock name
- Input: addressBlockRange of type **String** - The addressRange name
- Input: addressBlockWidth of type **String** - The addressWidth name

F.7.58.9 addBankSubspaceMap

Description: Adds a subspaceMap to a bank element.

- Returns: subSpaceMap of type **String** - Handle to the added subSpaceMap
- Input: bankID of type **String** - Handle to a bank element
- Input: initiatorRef of type **String** - Name of the referenced initiator busInterface

F.7.58.10 addLocalMemoryMapBank

Description: Adds a bank to the given localMemoryMap element

- Returns: bankID of type **String** - Handle to a new bank element
- Input: localMemoryMapID of type **String** - Handle to a localMemoryMap element
- Input: name of type **String** - Bank name
- Input: bankAlignment of type **String** - The bankAlignment attribute
- Input: baseAddress of type **String** - Bank base address
- Input: addressBlockName of type **String** - Name of the default addressBlock
- Input: addressBlockRange of type **String** - Range expression of the default addressBlock
- Input: addressBlockWidth of type **String** - Width expression of the default addressBlock

F.7.58.11 addMemoryMapAddressBlock

Description: Adds an addressBlock with the given name, baseAddress, range, and width to the given memoryMap element.

- Returns: addressBlockID of type **String** - Handle to a new addressBlock element
- Input: memoryMapID of type **String** - Handle to a memoryMap element
- Input: name of type **String** - AddressBlock name
- Input: baseAddress of type **String** - AddressBlock base address
- Input: range of type **String** - AddressBlock range
- Input: width of type **String** - AddressBlock width

F.7.58.12 addMemoryMapBank

Description: Adds a bank to the given memoryMap element

- Returns: bankID of type **String** - Handle to a new bank element
- Input: memoryMapID of type **String** - Handle to a memoryMap element
- Input: name of type **String** - Bank name
- Input: bankAlignment of type **String** - The bankAlignment attribute
- Input: baseAddress of type **String** - Bank base address
- Input: addressBlockName of type **String** - Name of the default addressBlock
- Input: addressBlockRange of type **String** - Range expression of the default addressBlock
- Input: addressBlockWidth of type **String** - Width expression of the default addressBlock

F.7.58.13 addMemoryMapMemoryRemap

Description: Adds a memoryRemap to memoryMap element.

- Returns: memoryRemapID of type **String** - Handle to a memoryRemap element
- Input: memoryMapID of type **String** - Handle to a memoryMap element
- Input: name of type **String** - Name of the memoryRemap
- Input: modeRef of type **String** - Name of the referenced mode
- Input: priority of type **Long** - Priority of the modeRef

F.7.58.14 addMemoryMapSubspaceMap

Description: Adds a subspaceMap with the given name, initiatorRef, and baseAddress to the given memory map.

- Returns: subspaceMapID of type **String** - Handle to a new subspaceMap element
- Input: memoryMapID of type **String** - Handle to a memoryMap element
- Input: name of type **String** - SubspaceMap name
- Input: initiatorRef of type **String** - SubspaceMap initiatorRef
- Input: baseAddress of type **String** - SubspaceMap base address

F.7.58.15 addMemoryRemapAddressBlock

Description: Adds an addressBlock on a memoryRemap element.

- Returns: addressBlockID of type **String** - Handle to the added addressBlock
- Input: memoryRemapID of type **String** - Handle to a memoryRemap element

- Input: name of type **String** - AddressBlock name
- Input: baseAddress of type **String** - AddressBlock base address
- Input: range of type **String** - AddressBlock range
- Input: width of type **String** - AddressBlock width

F.7.58.16 addMemoryRemapBank

Description: Adds a memoryRemap to a memoryRemap element.

- Returns: bankID of type **String** - Handle to the added bank
- Input: memoryRemapID of type **String** - Handle to a memoryMap or memoryRemap element
- Input: name of type **String** - Bank name
- Input: baseAddress of type **String** - Bank base address
- Input: bankAlignment of type **String** - Bank alignment

F.7.58.17 addMemoryRemapModeRef

Description: Adds a modeRef to a memoryRemap element.

- Returns: modeRefID of type **String** - the modeRef identifier
- Input: memoryRemapID of type **String** - Handle to a memoryRemap element
- Input: modeRef of type **String** - Reference to mode
- Input: priority of type **Long** - The priority of the modeRef

F.7.58.18 addMemoryRemapSubspaceMap

Description: Adds a subspaceMap to a memoryRemapID element.

- Returns: subSpaceMapID of type **String** - Handle to the added subSpaceMap
- Input: memoryRemapID of type **String** - Handle to a memoryRemap Element
- Input: name of type **String** - Name of a subspaceMap
- Input: initiatorRef of type **String** - Name of the referenced initiator busInterface
- Input: baseAddress of type **String** - Value of the baseAddress

F.7.58.19 addRegisterFileRegisterFile

Description: Adds a registerFile with the given name, addressOffset, and range to the given registerFile element.

- Returns: registerFileID of type **String** - Handle to the added registerFile
- Input: registerFileID of type **String** - Handle to a registerFile element
- Input: name of type **String** - RegisterFile name
- Input: addressOffset of type **String** - RegisterFile address offset
- Input: range of type **String** - RegisterFile range

F.7.58.20 removeAccessPolicyModeRef

Description: Removes the given modeRef from its containing accessPolicy element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element
- Input: modeRef of type **String** - Name of the referenced access mode

F.7.58.21 removeAddressBlockArray

Description: Removes the array on the addressBlock element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.58.22 removeAddressBlockRefIndex

Description: Removes the given index from its containing addressBlockRef element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indexID` of type ***String*** - Handle to the index element

F.7.58.23 removeAddressBlockRegister

Description: Removes the given register element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.58.24 removeAddressBlockRegisterFile

Description: Removes the given registerFile element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.58.25 removeAddressBlockTypeIdentifier

Description: Removes the typeIdentifier from the given addressBlock element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.58.26 removeAddressBlockUsage

Description: Removes the usage from the given addressBlock element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.58.27 removeAddressBlockVolatility

Description: Removes the volatility from the given addressBlock element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.58.28 removeAliasOfAddressBlockRef

Description: Removes an addressBlockRef on an aliasOf element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `aliasOfID` of type ***String*** - Handle to an aliasOf element

F.7.58.29 removeAliasOfAddressSpaceRef

Description: Removes the addressSpace Reference on an aliasOf element from registerField.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.58.30 removeAliasOfBankRef

Description: Removes the given bankRef from its containing aliasOf element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankrefID of type **String** - Handle to an bankRef element

F.7.58.31 removeAliasOfMemoryMapRef

Description: Removes the memoryMapRef on the aliasOf of a field on a register element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.58.32 removeBankAccessPolicy

Description: Removes the given accessPolicy from its containing bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessPolicyID of type **String** - Handle to an accessPolicy element

F.7.58.33 removeBankAddressBlock

Description: Removes an addressBlock with the given addressBlockID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.58.34 removeBankBank

Description: Removes the given bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankOrLocalBankID of type **String** - Handle to bank or localBank element

F.7.58.35 removeBankSubspaceMap

Description: Removes the given subspaceMap from its containing bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subspaceMapID of type **String** - Handle to a subspaceMap element

F.7.58.36 removeBankUsage

Description: Removes usage from the given bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankID of type **String** - Handle to bank

F.7.58.37 removeBankVolatility

Description: Removes volatility from the given bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankID of type **String** - Handle to a bank element

F.7.58.38 removeMemoryMapAddressBlock

Description: Removes the given addressBlock element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.58.39 removeMemoryMapBank

Description: Removes the given bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankOrLocalBankID of type **String** - Handle to bank or localBank element

F.7.58.40 removeMemoryMapMemoryRemap

Description: Removes the given memoryRemap from its containing memoryMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryRemapID of type **String** - Handle to a memoryRemap element

F.7.58.41 removeMemoryMapShared

Description: Removes the shared from the given memoryMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryMapID of type **String** - Handle to a memoryMap element

F.7.58.42 removeMemoryMapSubspaceMap

Description: Removes the given subspaceMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subSpaceMapID of type **String** - Handle to a subspaceMap element

F.7.58.43 removeMemoryRemapBank

Description: Removes the given bank from a memoryRemap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankID of type **String** - Handle to a bank element

F.7.58.44 removeMemoryRemapMapAddressBlock

Description: Removes the given addressBlock from its containing memoryRemap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockID of type **String** - Handle to an addressBlock element

F.7.58.45 removeMemoryRemapModeRef

Description: Removes the given modeRef from its containing memoryRemap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeRefID of type **String** - Handle to a modeRef element

F.7.58.46 removeMemoryRemapSubspaceMap

Description: Removes the given subspaceMap from its containing memoryRemap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subspaceMapID of type **String** - Handle to subspaceMap element

F.7.58.47 removeRegisterArray

Description: Removes the array on the given register.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerID of type **String** - Handle to a register element

F.7.58.48 removeRegisterFileArray

Description: Removes the array on the register.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileID of type **String** - Handle to a registerFile element

F.7.58.49 setAddressBlockAddressBlockDefinitionRef

Description: Sets the addressBlockDefinitionRef with his value and typeDefinitions.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockID of type **String** - Handle to an addressBlock element
- Input: value of type **String** - Name of the referenced addressBlockDefinition in an external typeDefinitions
- Input: typeDefinitions of type **String** - Name of the component externalTypeDefinitions

F.7.58.50 setAddressBlockArray

Description: Sets the array on the addressBlock element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockID of type **String** - Handle to an addressBlock element
- Input: dim of type **String** - The dim value

F.7.58.51 setAddressBlockBaseAddress

Description: Sets the baseAddress value of the given addressBlock

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockID of type **String** - Handle to an addressBlock
- Input: baseAddress of type **String** - Expression of the base address

F.7.58.52 setAddressBlockDefinitionRef

Description: Sets the value field of the given addressblock definition ref of an addressblock.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `addressBlockDefinitionRefID` of type **String** - Handle to an `addressBlockDefinitionRef`
- Input: `value` of type **String** - Handle to the new value

F.7.58.53 setAddressBlockRange

Description: Sets the range with the given value for the given addressBlock element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type **String** - Handle to an `addressBlock` element
- Input: `range` of type **String** - The `addressBlock` range

F.7.58.54 setAddressBlockTypeIdentifier

Description: Sets the typeIdentifier with the given value for the given addressBlock element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type **String** - Handle to an `addressBlock` element
- Input: `typeIdentifier` of type **String** - The `addressBlock` typeIdentifier

F.7.58.55 setAddressBlockUsage

Description: Sets the usage with the given value for the given addressBlock element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type **String** - Handle to an `addressBlock` element
- Input: `usage` of type **String** - The `addressBlock` usage

F.7.58.56 setAddressBlockVolatility

Description: Sets the volatility with the given value for the given addressBlock element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type **String** - Handle to an `addressBlock` element
- Input: `_volatile` of type **Boolean** - The `addressBlock` volatility

F.7.58.57 setAddressBlockWidth

Description: Sets the width of the given addressBlock element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type **String** - Handle to an `addressBlock` element
- Input: `width` of type **String** - Width expression to be set

F.7.58.58 setAliasOfAddressBlockRef

Description: Sets an `addressBlockRef` on an `aliasOf` element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `aliasOfID` of type **String** - Handle to an `aliasOf` element

- Input: addressBlockRef of type **String** - Name of the referenced addressBlock

F.7.58.59 setAliasOfAddressSpaceRef

Description: Sets the addressSpace Reference on the given aliasOf element from registerField.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle to an aliasOf element
- Input: addressSpaceRef of type **String** - Name of the referenced addressSpace

F.7.58.60 setAliasOfMemoryMapRef

Description: Sets the memoryMapRef on the aliasOf of a field on a register element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle to an aliasOf element
- Input: memoryMapRef of type **String** - Name of the referenced memoryMap

F.7.58.61 setAliasOfMemoryRemapRef

Description: Sets an memoryRemapRef on an aliasOf element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle to an aliasOf element
- Input: memoryRemapRef of type **String** - Name of the referenced memoryRemap

F.7.58.62 setBankBankDefinitionRef

Description: Sets the bankDefinitionRef on a bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankID of type **String** - Handle to a bank element
- Input: value of type **String** - Name of the referenced bankFileDefinition in an externalTypeDefinitions
- Input: typeDefinitions of type **String** - Name of the component externalTypeDefinitions

F.7.58.63 setBankBaseAddress

Description: Sets the baseAddress value of the given bank and returns an identifier to an addressBlock.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankID of type **String** - Handle to an bank
- Input: baseAddress of type **String** - Expression of the base address

F.7.58.64 setBankUsage

Description: Sets usage with the given value for the given bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankID of type **String** - Handle to bank or localBank element
- Input: usage of type **String** - Bank usage

F.7.58.65 setBankVolatility

Description: Sets volatility with the given value for the given bank element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankOrLocalBankID of type **String** - Handle to bank or localBank element
- Input: volatile of type **Boolean** - Bank volatility

F.7.58.66 setBroadcastToAddressSpaceRef

Description: Sets the addressSpace Reference on the given broadcastTo element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to an broadcastTo element
- Input: addressSpaceRef of type **String** - Name of the referenced addressSpace

F.7.58.67 setMemoryMapMemoryMapDefinitionRef

Description: Sets a Value to MemoryMapDefintionRef.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryMapID of type **String** - Handle to a memoryMap
- Input: value of type **String** - Name of the referenced memoryMapDefinition in an externalTypeDefinitions
- Input: typeDefinitions of type **String** - Name of the component externalTypeDefinitions

F.7.58.68 setMemoryMapShared

Description: Sets the shared flag with the given value to the given memoryMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryMapID of type **String** - Handle to a memoryMap element
- Input: shared of type **String** - MemoryMap shared enumeration value

F.7.58.69 setMemoryRemapRemapDefinitionRef

Description: Sets the remapDefinitionRef on a memoryRemap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryRemapID of type **String** - Handle to a memoryMap element
- Input: value of type **String** - Name of the referenced remapDefinition in an externalTypeDefinitions
- Input: typeDefinitions of type **String** - Name of the component externalTypeDefinitions

F.7.58.70 setRegisterArray

Description: Sets the array on the register.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerID of type **String** - Handle to a register element
- Input: dimValue of type **String** - the dim value to set on the array

F.7.58.71 setRegisterFieldArray

Description: Sets the array on the given field of a register.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFieldID of type **String** - Handle to a field element on a register

- Input: value of type **String** - Dimension of the array

F.7.58.72 setRegisterFieldBitOffset

Description: Sets the bitOffset expression on the given field of a register.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFieldID of type **String** - Handle to a field element on a register
- Input: bitOffset of type **String** - The bitOffset expression

F.7.58.73 setRegisterFileArray

Description: Sets the array on the register.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileID of type **String** - Handle to a registerFile element
- Input: dimValue of type **String** - The dim value to set on the array

F.7.58.74 setSubSpaceMapBaseAddress

Description: Sets the baseAddress value of the given subSpaceMap

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subSpaceMapID of type **String** - Handle to an subSpaceMap
- Input: baseAddress of type **String** - Expression of the base address

F.7.59 Miscellaneous (BASE)

F.7.59.1 getArgumentValue

Description: Returns the value defined on the given fileSet function argument element.

- Returns: value of type **String** - The value of the value element
- Input: argumentID of type **String** - Handle to an argument element

F.7.59.2 getArgumentValueExpression

Description: Returns the expression of the value defined on the given fileSet function argument element.

- Returns: expression of type **String** - The expression of the value element
- Input: argumentID of type **String** - Handle to an argument element

F.7.59.3 getArgumentValueID

Description: Returns the handle to the value defined on the given fileSet function argument element.

- Returns: valueID of type **String** - Handle to the value element
- Input: argumentID of type **String** - Handle to an argument element

F.7.59.4 getBooleanValue

Description: Returns the boolean value of the given element.

- Returns: value of type **Boolean** - The value of the given element
- Input: elementID of type **String** - Handle to an IP-XACT element

F.7.59.5 getDefineValue

Description: Returns the value defined on the given fileSet define element.

- Returns: value of type **String** - The value of the value element
- Input: defineID of type **String** - Handle to a define element

F.7.59.6 getDefineValueExpression

Description: Returns the expression on a define element.

- Returns: expression of type **String** - The expression of the value element
- Input: defineID of type **String** - Handle to a define element

F.7.59.7 getDefineValueID

Description: Returns the handle to the value defined on the given fileSet define element.

- Returns: valueID of type **String** - Handle to the value element
- Input: defineID of type **String** - Handle to a define element

F.7.59.8 getExpression

Description: Returns the expression defined on the given expressionContainer element.

- Returns: expression of type **String** - The expression (not evaluated)
- Input: expressionID of type **String** - Handle to an expression element

F.7.59.9 getExpressionIntValue

Description: Returns the value of the given expression.

- Returns: expression of type **Long** - The value of the expression (not evaluated)
- Input: expressionID of type **String** - Handle to an expression

F.7.59.10 getExpressionValue

Description: Returns the value of the given expression.

- Returns: expression of type **String** - The value of the expression (evaluated)
- Input: expressionID of type **String** - Handle to an expression element

F.7.59.11 getGroup

Description: Returns the name of the given group element.

- Returns: name of type **String** - The group name
- Input: groupID of type **String** - Handle to a group element

F.7.59.12 getPartSelectIndexIDs

Description: Returns the handles to all the indexes defined on the given partSelect element.

- Returns: indexID of type **String** - List of handles to the index elements
- Input: partSelectID of type **String** - Handle to a partSelect element

F.7.59.13 getPartSelectIndices

Description: Returns all the indices values defined on the given partSelect element.

- Returns: `indices` of type **Long** - List of the indices values
- Input: `partSelectID` of type **String** - Handle to a partSelect

F.7.59.14 getPartSelectIndicesExpression

Description: Returns all the indices expressions defined on the given partSelect element.

- Returns: `indicesExpression` of type **String** - List of the indices expressions
- Input: `partSelectID` of type **String** - Handle to a partSelect

F.7.59.15 getPartSelectRange

Description: Returns the range (resolved) value defined on the given partSelect element.

- Returns: `range` of type **Long** - Array of two range values: left and right
- Input: `partSelectID` of type **String** - Handle to a partSelect element

F.7.59.16 getPartSelectRangeExpression

Description: Returns the range expression defined on the given partSelect element.

- Returns: `range` of type **String** - Array of two range expressions: left and right
- Input: `partSelectID` of type **String** - Handle to a partSelect element

F.7.59.17 getPartSelectRangeLeftID

Description: Returns the handle to the left range defined on the given partSelect element.

- Returns: `leftRangeID` of type **String** - Handle to the left range
- Input: `partSelectID` of type **String** - Handle to a partSelect element

F.7.59.18 getPartSelectRangeRightID

Description: Returns the handle to the right range defined on the given partSelect element.

- Returns: `rightRangeID` of type **String** - Handle to the right range
- Input: `partSelectID` of type **String** - Handle to a partSelect element

F.7.59.19 getValue

Description: Returns the string value of the given element.

- Returns: `value` of type **String** - The value of the given element
- Input: `elementID` of type **String** - Handle to an IP-XACT element

F.7.59.20 getXML

Description: Returns the XML fragment.

- Returns: `xmlString` of type **String** - The XML fragment
- Input: `elementID` of type **String** - IP-XACT XML element

F.7.60 Miscellaneous (EXTENDED)

F.7.60.1 addPartSelectIndex

Description: Adds an index on a partSelect element.

- Returns: indexID of type **String** - the index identifier
- Input: partSelectID of type **String** - Handle to a partSelect
- Input: value of type **String** - the index value

F.7.60.2 end

Description: Terminates the connection to the DE.

- Returns: status of type **Long** - Status indicator from the DE. Non-Zero implies an error
- Input: genStatus of type **Long** - Status indicator from the generator. Non-zero implies an error
- Input: message of type **String** - Message that the DE may display to the user

F.7.60.3 init

Description: API initialization function. Must be called before any other TGI call.

- Returns: status of type **Boolean** - True if the call is successful False otherwise
- Input: apiVersion of type **String** - The API version with which the generator is defined to work
- Input: failureMode of type **String** - Compatibility failure mode (fail, error or warning)
- Input: message of type **String** - Message that the DE may display to the user

F.7.60.4 isSetElement

Description: Checks if the given element is set or not.

- Returns: isSet of type **Boolean** - True if the element is defined; False otherwise
- Input: elementContainerID of type **String** - Handle to the parent of the element to check
- Input: elementName of type **String** - Name of the element

F.7.60.5 message

Description: Sends the given message level and message text to the DE.

- Returns: status of type **Boolean** - True if the call is successful, False otherwise
- Input: severity of type **String** - message level
- Input: message of type **String** - message text

F.7.60.6 registerCatalogVLMVs

Description: Registers all the VLMVs defined in the given catalog.

- Returns: status of type **Boolean** - True if the call is successful, False otherwise
- Input: catalogID of type **String** - Catalog who's VLMVs are to be registered

F.7.60.7 registerVLMV

Description: Registers the VLMV contained in the given file, possibly replacing an existing VLMV.

- Returns: status of type **Boolean** - True if the call is successful, False otherwise
- Input: fileName of type **String** - Location of file that is to be registered

- Input: `replace` of type **Boolean** - Replace existing VLNV with new content

F.7.60.8 removePartSelectIndex

Description: Removes the given index from its containing a partSelect indices element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `indexID` of type **String** - The identifier of an index element

F.7.60.9 resolveExpression

Description: Resolves any expression for given parameter and value pairs.

- Returns: `expression` of type **String** - The resolved expression
- Input: `expression` of type **String** - The expression to solve
- Input: `values` of type **String[]** - Name and value pairs formatted as name = value, where value must be a constant.

F.7.60.10 save

Description: Save all edits done in generator to DE; document must be valid.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)

F.7.60.11 setArgumentValue

Description: Sets the value on an argument element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `argumentID` of type **String** - Handle to an argument element
- Input: `value` of type **String** - The value to set on the argument

F.7.60.12 setBooleanValue

Description: Sets the boolean value on the given element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type **String** - Handle to an IP-XACT element
- Input: `value` of type **Boolean** - The value to be set

F.7.60.13 setDefineValue

Description: Sets the value on a define element.

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `argumentID` of type **String** - Handle to an argument element
- Input: `value` of type **String** - The value to set on the argument

F.7.60.14 setExpressionValue

Description: Sets the value to the given expression

- Returns: `status` of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: `expressionID` of type **String** - Handle to an expression
- Input: `value` of type **String** - New value expression.

F.7.60.15 setPartSelectRange

Description: Sets the PartSelect Range field.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `partSelectID` of type ***String*** - Handle to a partSelect element
- Input: `left` of type ***String*** - Left value of the range
- Input: `right` of type ***String*** - Right value of the range

F.7.60.16 setValue

Description: Sets the string value on the given element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an IP-XACT element
- Input: `value` of type ***String*** - The value to be set

F.7.60.17 removePartSelectRange

Description: Remove the range of a partSelect element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `partSelectID` of type ***String*** - Handle to partSelect

F.7.60.18 unregisterCatalogVLNVs

Description: Unregister all the VLNVs in the given catalog.

- Returns: `status` of type ***Boolean*** - True if the call is successful, False otherwise
- Input: `catalogID` of type ***String*** - Catalog which VLNVs are to be unregistered

F.7.60.19 unregisterVLNV

Description: Unregisters the given VLNV.

- Returns: `status` of type ***Boolean*** - True if the call is successful, False otherwise
- Input: `VLNV` of type ***String[]*** - VLNV that is to be unregistered

F.7.61 Module parameter (BASE)

F.7.61.1 getModuleParameterIDs

Description: Returns the handles to all the module or type parameters defined on the given element.

- Returns: `moduleOrTypeParameterIDs` of type ***String*** - List of handles to module or type parameter elements
- Input: `moduleOrTypeParameterContainerElementID` of type ***String*** - Handle to an element that has moduleOrTypeParameter elements

F.7.61.2 getModuleParameterValue

Description: Returns the value defined on the given module or type parameter element.

- Returns: `value` of type ***String*** - The parameter value
- Input: `moduleParameterID` of type ***String*** - Handle to a module or type parameter element

F.7.61.3 getModuleParameterValueExpression

Description: Returns expression defined on the given module or type parameter element.

- Returns: `expression` of type ***String*** - The parameter expression
- Input: `moduleParameterID` of type ***String*** - Handle to a module or type parameter element

F.7.62 Module parameter (EXTENDED)

F.7.62.1 addModuleParameter

Description: Adds a `moduleOrTypeParameter` with the given name and given value to the given element.

- Returns: `moduleOrTypeParameterID` of type ***String*** - Handle to a new `moduleOrTypeParameter`
- Input: `moduleOrTypeParameterContainerElementID` of type ***String*** - Handle to an element that has `moduleOrTypeParameter` elements
- Input: `name` of type ***String*** - The `moduleOrTypeParameter` name
- Input: `expression` of type ***String*** - The `moduleOrTypeParameter` expression

F.7.62.2 removeModuleParameter

Description: Removes the given `moduleOrTypeParameter`.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `moduleParameterID` of type ***String*** - Handle to a `moduleOrTypeParameter` element

F.7.62.3 setModuleParameterValue

Description: Sets the value of the given `moduleOrTypeParameter`.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a `moduleOrTypeParameter` element
- Input: `expression` of type ***String*** - `moduleOrTypeParameter` expression

F.7.63 Name group (BASE)

F.7.63.1 getDescription

Description: Returns the description defined on the given element.

- Returns: `description` of type ***String*** - The description of the given element
- Input: `elementID` of type ***String*** - Handle to an element that has a `nameGroup`

F.7.63.2 getDisplayName

Description: Returns the display name defined on the given element.

- Returns: `displayName` of type ***String*** - The display name of the given element
- Input: `elementID` of type ***String*** - Handle to an element that has a `nameGroup`

F.7.63.3 getName

Description: Returns the name defined on the given element.

- Returns: name of type **String** - The name of the given element
- Input: elementID of type **String** - Handle to an element that has a nameGroup

F.7.63.4 getShortDescription

Description: Returns the short description of the given element. (Unresolved).

- Returns: shortDescription of type **String** - The short description
- Input: elementID of type **String** - Handle to a name group element

F.7.64 Name group (EXTENDED)

F.7.64.1 removeDescription

Description: Removes the given description from its containing described element; fails if description is not already set.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has a nameGroup

F.7.64.2 removeDisplayName

Description: Removes the given displayName from its containing groupName element; fails if displayName is not already set.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has a nameGroup

F.7.64.3 removeName

Description: Removes the name of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has a nameGroup

F.7.64.4 removeShortDescription

Description: Removes the given shortDescription from its containing groupName element; fails if shortDescription is not already set.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has a nameGroup

F.7.64.5 setDescription

Description: Sets given description for the given element; fails if description is not already set.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has a nameGroup
- Input: description of type **String** - New description

F.7.64.6 setDisplayName

Description: Sets given display name for the given element; fails if display name is not already set.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: elementID of type **String** - Handle to an element that has a nameGroup
- Input: displayName of type **String** - New display name

F.7.64.7 setName

Description: Sets the name of the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to an element that has a nameGroup
- Input: name of type **String** - New name

F.7.64.8 setShortDescription

Description: Sets the short description.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: elementID of type **String** - Handle to a nameGroup
- Input: shortDescription of type **String** - Handle to the new value

F.7.65 Parameter (BASE)

F.7.65.1 getModuleParameterDataTypeDefinitionRefByID

Description: Returns the handle to the file referenced from the given moduleParameterType element.

- Returns: fileID of type **String** - Handle to the referenced file element
- Input: moduleParameterTypeID of type **String** - Handle to a moduleParameterType element

F.7.65.2 getParameterChoiceRefByName

Description: Returns the choice referenced from the given parameter element.

- Returns: choiceRef of type **String** - The referenced choice
- Input: parameterBaseTypeID of type **String** - Handle to a parameter element

F.7.65.3 getParameterIDFromReferenceID

Description: Returns the handle to the parameter element from the given component and reference elements.

- Returns: parameterID of type **String** - Handle to the parameter element
- Input: parametrizedID of type **String** - Handle to an element containing parameters
- Input: referenceID of type **String** - Handle to a referenceID element

F.7.65.4 getParameterIDs

Description: Returns the handles to all the parameters defined on the given element.

- Returns: parameterIDs of type **String** - List of handles to parameter elements
- Input: parameterContainerElementID of type **String** - Handle to an element that has parameter elements

F.7.65.5 getParameterNameFromReferenceID

Description: Returns the parameter name from the given component and reference elements.

- Returns: parameterName of type **String** - The parameter name

- Input: parametrizedID of type **String** - Handle to an element containing parameters
- Input: referenceID of type **String** - Handle to a referenceID element

F.7.65.6 getParameterValue

Description: Returns the value defined on the given parameter element.

- Returns: value of type **String** - The parameter value
- Input: parameterID of type **String** - Handle to a parameter element

F.7.65.7 getParameterValueExpression

Description: Returns the expression defined on the given parameter element.

- Returns: expression of type **String** - The parameter expression
- Input: parameterID of type **String** - Handle to a parameter element

F.7.65.8 getParameterValueID

Description: Returns the handle expression of the given parameter element.

- Returns: parameterValueID of type **String** - Handle to the parameterValue element
- Input: parameterID of type **String** - Handle to a parameter element

F.7.66 Parameter (EXTENDED)

F.7.66.1 addParameter

Description: Adds a parameter with the given name and given value to the given element.

- Returns: parameterID of type **String** - Handle to a new parameter
- Input: parameterContainerElementID of type **String** - Handle to an element that has parameter elements
- Input: name of type **String** - Parameter name
- Input: expression of type **String** - Parameter expression

F.7.66.2 removeConfigGroupsAttribute

Description: Removes a configGroups of the given parameter.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: parameterID of type **String** - Handle to a parameter

F.7.66.3 removeParameter

Description: Removes the given parameter.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: parameterID of type **String** - Handle to a parameter element

F.7.66.4 setParameterValue

Description: Sets the value of the given parameter.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: parameterID of type **String** - Handle to a parameter element

- Input: expression of type **String** - Parameter expression

F.7.67 Port (BASE)

F.7.67.1 getAccessPortAccessType

Description: Returns the accessType defined on the given access element.

- Returns: accessType of type **String** - The port access type
- Input: accessID of type **String** - Handle to an access element

F.7.67.2 getDomainTypeDefTypeDefinitionIDs

Description: Returns the handles to all the typeDefinitions defined on the given domainTypeDef element.

- Returns: typeDefinitionIDs of type **String** - List of handles to the typeDefinition elements
- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element

F.7.67.3 getDomainTypeDefTypeDefinitions

Description: Returns the typeDefinitions defined on the given domainTypeDef element.

- Returns: typeDefinitions of type **String** - List of typeDefinitions
- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element

F.7.67.4 getDomainTypeDefTypeName

Description: Returns the typeName defined on the given domainTypeDef element.

- Returns: typeName of type **String** - The type name
- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element

F.7.67.5 getDomainTypeDefTypeNameID

Description: Returns the handle to the typeName defined on the given domainTypeDef element.

- Returns: typeNameID of type **String** - Handle to the typeName element
- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element

F.7.67.6 getDomainTypeDefViewIDs

Description: Returns the handles to all the views defined on the given domainTypeDef element.

- Returns: viewIDs of type **String** - List of handles to the view elements
- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element

F.7.67.7 getDomainTypeDefViewRefsIDs

Description: Returns the handles to all the viewRefs defined on the given domainTypeDef element.

- Returns: viewRefIDs of type **String** - List of handles to the viewRef elements
- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element

F.7.67.8 getDomainTypeDefViewRefs

Description: Returns all the viewRefs defined on the given domainTypeDef element.

- Returns: `viewRefs` of type ***String*** - List of the referenced views
- Input: `domainTypeDefID` of type ***String*** - Handle to a `domainTypeDef` element

F.7.67.9 getFieldDefinitionAccessPoliciesIDs

Description: Returns the handles to all the `AccessPolicies` defined on the given `fieldDefinition` element.

- Returns: `accessPoliciesIDs` of type ***String*** - List of handles to the `accessPolicies` elements
- Input: `fieldDefinitionID` of type ***String*** - Handle to a `fieldDefinition` element

F.7.67.10 getFieldMapFieldSliceID

Description: Returns the handle to `fieldSlice` defined on the given `fieldMap` element.

- Returns: `fieldSliceID` of type ***String*** - Handle to the `fieldSlice` element
- Input: `fieldMapID` of type ***String*** - Handle to a `fieldMap` element

F.7.67.11 getFieldMapModeRefByID

Description: Returns the handle to the mode referenced from the given `fieldMap` element.

- Returns: `modeID` of type ***String*** - Handle to the referenced mode element
- Input: `fieldMapID` of type ***String*** - Handle to a `fieldMap` element
- Input: `modeRef` of type ***String*** - The referenced mode

F.7.67.12 getFieldMapModeRefByNames

Description: Returns all the `modeRefs` defined on the given `fieldMap` element.

- Returns: `modeRefs` of type ***String*** - List of the referenced modes
- Input: `fieldMapID` of type ***String*** - Handle to a `fieldMap` element

F.7.67.13 getFieldMapModeRefIDs

Description: Returns the handles to all the `modeRef` defined on the given `fieldMap` element

- Returns: `modeRefIDs` of type ***String*** - Handles to `modeRefs` elements
- Input: `fieldMapID` of type ***String*** - Handle to a `fieldMap` element

F.7.67.14 getFieldMapModeRefs

Description: Returns all the `modeRefs` defined on the given `fieldMap` element.

- Returns: `modeRefs` of type ***String*** - List of the referenced modes
- Input: `fieldMapID` of type ***String*** - Handle to a `fieldMap` element

F.7.67.15 getFieldMapPartSelectID

Description: Returns the handle to the `partSelect` defined on the given `fieldMap` element.

- Returns: `partSelectID` of type ***String*** - Handle to the `partSelect` element
- Input: `fieldMapID` of type ***String*** - Handle to a `fieldMap` element

F.7.67.16 getFieldMapSubPortReferenceIDs

Description: Returns the handles to all the `subPortReferences` defined on the given `fieldMap` element.

- Returns: `subPortReferenceID` of type ***String*** - List of handles to the `subPortReference` elements
- Input: `fieldMapID` of type ***String*** - Handle to a `fieldMap` element

F.7.67.17 getPayloadExtension

Description: Returns the name of the payload extension defined on the given payload element.

- Returns: `extension` of type ***String*** - The payload extension name
- Input: `payloadID` of type ***String*** - Handle to a protocol payload element

F.7.67.18 getPayloadExtensionID

Description: Returns the handle to the payload extension defined on the given payload element.

- Returns: `extensionID` of type ***String*** - Handle to the payload extension element
- Input: `payloadID` of type ***String*** - Handle to a protocol payload element

F.7.67.19 getPayloadType

Description: Returns the type defined on the given payload element.

- Returns: `type` of type ***String*** - The payload type
- Input: `payloadID` of type ***String*** - Handle to a payload element

F.7.67.20 getPortAccessID

Description: Returns the handle to the access defined on the given port element.

- Returns: `portAccessID` of type ***String*** - Handle to the `portAccess` element
- Input: `portID` of type ***String*** - Handle to a port element

F.7.67.21 getPortDomainTypeDefIDs

Description: Returns the handles to all the `domainTypeDef` defined on the given port wire element.

- Returns: `domainTypeDefIDs` of type ***String*** - List of handles to `domainTypeDef` elements
- Input: `wireID` of type ***String*** - Handle to a port wire element

F.7.67.22 getPortFieldMapIDs

Description: Returns the handles to all the `fieldMap` defined on the given port element.

- Returns: `fieldMapIDs` of type ***String*** - List of handles to `fieldMap` elements
- Input: `portID` of type ***String*** - Handle to a mode element

F.7.67.23 getPortSignalTypeDefIDs

Description: Returns the handles to all the `signalTypeDef` defined on the given port wire element.

- Returns: `signalTypeDefIDs` of type ***String*** - List of handles to `signalTypeDef` elements
- Input: `wireID` of type ***String*** - Handle to a port wire element

F.7.67.24 getPortStructuredID

Description: Returns the handle to the structured element defined on the given port element.

- Returns: `structuredID` of type ***String*** - Handle to the structured element

- Input: `portID` of type ***String*** - Handle to a port element

F.7.67.25 getPortStructuredInterfaceID

Description: Returns the handle to the interface

- Returns: `interfaceID` of type ***String*** - Handle to an interface element
- Input: `structuredID` of type ***String*** - Handle to a port structured element

F.7.67.26 getPortStructuredStructID

Description: Returns the handle to the struct defined on the given structuredPort element.

- Returns: `structID` of type ***String*** - Handle to the struct element
- Input: `structuredID` of type ***String*** - Handle to a port structured

F.7.67.27 getPortStructuredStructPortTypeDefIDs

Description: Returns the handles to all the structPortTypeDef typeDefinitions defined on the given port structured element.

- Returns: `structPortTypeDefIDs` of type ***String*** - List of handles to the structPortTypeDef elements
- Input: `structuredID` of type ***String*** - Handle to a port structured element

F.7.67.28 getPortStructuredSubPortIDs

Description: Returns the handles to all the subPorts defined on the given port structured element.

- Returns: `subPortIDs` of type ***String*** - List of handles to the subPort elements
- Input: `structuredID` of type ***String*** - Handle to a port structured element

F.7.67.29 getPortStructuredUnionID

Description: Returns the handle to the union element defined on the given structuredPort element.

- Returns: `unionID` of type ***String*** - Handle to the union element
- Input: `structuredID` of type ***String*** - Handle to a port structured element

F.7.67.30 getPortStructuredVectorIDs

Description: Returns the handles to all the vectors defined on the given port structured element.

- Returns: `vectorIDs` of type ***String*** - List of handles to the vector elements
- Input: `structuredID` of type ***String*** - Handle to a port structured element

F.7.67.31 getPortStyle

Description: Returns the style (wire, transactional or structured) of the given port element.

- Returns: `style` of type ***String*** - The port style (wire or transactional or structured)
- Input: `portID` of type ***String*** - Handle to a port element

F.7.67.32 getPortTransactionalBusWidth

Description: Returns the busWidth value defined on the given port element.

- Returns: `width` of type ***Long*** - The busWidth value

- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.33 `getPortTransactionalBusWidthExpression`

Description: Returns the busWidth expression defined on the given port element.

- Returns: `widthExpression` of type ***String*** - The busWidth expression
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.34 `getPortTransactionalBusWidthID`

Description: Returns the handle to the busWidth defined on the given port transactional element.

- Returns: `widthID` of type ***String*** - Handle to the busWidth element
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.35 `getPortTransactionalID`

Description: Returns the handle to the transactional element defined on the given port element.

- Returns: `transactionalID` of type ***String*** - Handle to the transactional element
- Input: `portID` of type ***String*** - Handle to a port element

F.7.67.36 `getPortTransactionalInitiative`

Description: Returns the initiative defined on the given port transactional element.

- Returns: `initiative` of type ***String*** - The port initiative
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.37 `getPortTransactionalKind`

Description: Returns the kind defined on the given port transactional element.

- Returns: `kind` of type ***String*** - The port kind
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.38 `getPortTransactionalKindID`

Description: Returns the handle to the kind defined on the given port transactional element.

- Returns: `kind` of type ***String*** - Handle to the kind element
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.39 `getPortTransactionalMaxConnections`

Description: Returns the maxConnections defined on the given port element.

- Returns: `value` of type ***Long*** - The maximum number of connections allowed on the port
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.40 `getPortTransactionalMaxConnectionsExpression`

Description: Returns the maxConnections expression defined on the given port element.

- Returns: `valueExpression` of type ***String*** - The maxConnections expression
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.41 getPortTransactionalMaxConnectionsID

Description: Returns the handle to the maxConnections defined on the given port transactional element.

- Returns: `maxConnectionsID` of type ***String*** - Handle to the maximum number of connections allowed on this port
- Input: `portID` of type ***String*** - Handle to a port transactional element

F.7.67.42 getPortTransactionalMinConnections

Description: Returns the minConnections defined on the given port element.

- Returns: `value` of type ***Long*** - The minimum number of connections allowed on the port
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.43 getPortTransactionalMinConnectionsExpression

Description: Returns the minConnections expression defined on the given port element.

- Returns: `valueExpression` of type ***String*** - The minConnections expression
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.44 getPortTransactionalMinConnectionsID

Description: Returns the handle to the minConnections defined on the given port transactional element.

- Returns: `minConnectionsID` of type ***String*** - Handle to the minimum number of connections required on this port
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.45 getPortTransactionalProtocolID

Description: Returns the handle to the protocol defined on the given port transactional element.

- Returns: `ProtocolID` of type ***String*** - Handle to the protocol element
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.46 getPortTransactionalQualifierID

Description: Returns the handle to the qualifier defined on the given component port transactional element.

- Returns: `qualifierID` of type ***String*** - Handle to the qualifier element
- Input: `transactionalID` of type ***String*** - Handle to port transactional

F.7.67.47 getPortTransactionalTransTypeDefIDs

Description: Returns the handles to all the transTypeDefs defined on the given port element.

- Returns: `transTypeDefIDs` of type ***String*** - List of handles to transTypeDef elements
- Input: `transactionalID` of type ***String*** - Handle to a port transactional element

F.7.67.48 getPortWireConstraintSetIDs

Description: Returns the handles to all the constraintSet elements defined on given port wire element.

- Returns: `constraintSetIDs` of type ***String*** - List of handles to the constraintSet elements
- Input: `wireID` of type ***String*** - Handle to a port wire element

F.7.67.49 getPortWireDirection

Description: Returns the direction defined on the given port wire element.

- Returns: `direction` of type ***String*** - The port direction
- Input: `wireID` of type ***String*** - Handle to a port wire element

F.7.67.50 getPortWireDriverIDs

Description: Returns the handles to all the drivers defined on the given port wire element.

- Returns: `driverIDs` of type ***String*** - List of handles to the driver elements
- Input: `wireID` of type ***String*** - Handle to a port wire element

F.7.67.51 getPortWireID

Description: Returns the handle to wire element defined on the given port element.

- Returns: `wireID` of type ***String*** - Handle to the wire element
- Input: `portID` of type ***String*** - Handle to a port element

F.7.67.52 getPortWireQualifierID

Description: Returns the handle to the qualifier defined on the given component port wire element.

- Returns: `qualifierID` of type ***String*** - Handle to the qualifier element
- Input: `wireID` of type ***String*** - Handle to port wire element

F.7.67.53 getPortWireTypeDefIDs

Description: Returns the handles to all the wireTypeDefs defined on the given port element.

- Returns: `wireTypeDefIDs` of type ***String*** - List of handles to wireTypeDef elements
- Input: `wireID` of type ***String*** - Handle to a port wire element

F.7.67.54 getProtocolPayloadID

Description: Returns the handle to the payload defined on the given transactional port protocol element.

- Returns: `payloadID` of type ***String*** - Handle to a payload element
- Input: `protocolID` of type ***String*** - Handle to a protocol element

F.7.67.55 getProtocolProtocolType

Description: Returns the protocolType defined on the given protocol element.

- Returns: `protocolType` of type ***String*** - The protocol type
- Input: `protocolID` of type ***String*** - Handle to protocol element

F.7.67.56 getProtocolProtocolTypeID

Description: Returns the handle to the protocolType defined on the given protocol element.

- Returns: `protocolTypeID` of type ***String*** - Handle of protocol type
- Input: `protocolID` of type ***String*** - Handle to protocol element

F.7.67.57 getQualifierIsAddress

Description: Returns the isAddress boolean value of the given port qualifier element.

- Returns: `isAddress` of type **Boolean** - True if the isAddress qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.58 getQualifierIsClock

Description: Returns the isClock boolean value of the given port qualifier element.

- Returns: `isClock` of type **Boolean** - True if the isClock qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.59 getQualifierIsClockEn

Description: Returns the isClockEn boolean value of the given port qualifier element.

- Returns: `isClockEn` of type **Boolean** - True if the isClockEn qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.60 getQualifierIsClockEnID

Description: Returns the handle to the isClockEn element defined on the given port qualifier element.

- Returns: `isClockEnID` of type **String** - Handle to the isClockEn element
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.61 getQualifierIsData

Description: Returns the isData boolean value of the given port qualifier element.

- Returns: `isData` of type **Boolean** - True if the isData qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.62 getQualifierIsFlowControl

Description: Returns the isFlowControl boolean value of the given port qualifier element.

- Returns: `isFlowControl` of type **Boolean** - True if the isFlowControl qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.63 getQualifierIsFlowControlID

Description: Returns the handle to the isFlowControl element defined on the given port qualifier element.

- Returns: `isFlowControlID` of type **String** - Handle to the isFlowControl element
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.64 getQualifierIsInterrupt

Description: Returns the isInterrupt boolean value of the given port qualifier element.

- Returns: `isInterrupt` of type **Boolean** - True if the isInterrupt qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.65 getQualifierIsOpcode

Description: Returns the isOpCode boolean value of the given port qualifier element.

- Returns: `isOpCode` of type **Boolean** - True if the isOpCode qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.66 getQualifierIsPowerEn

Description: Returns the isPowerEn boolean value of the given port qualifier element.

- Returns: `isPowerEn` of type **Boolean** - True if the isPowerEn qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.67 getQualifierIsPowerEnID

Description: Returns the handle to the isPowerEn element defined on the given port qualifier element.

- Returns: `isPowerEnID` of type **String** - Handle to the isPowerEn element
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.68 getQualifierIsPowerEnPowerDomainRefByName

Description: Returns the powerDomain referenced from the given isPowerEn element.

- Returns: `powerDomainRef` of type **String** - The referenced powerDomain
- Input: `powerEnID` of type **String** - Handle to a powerEn element

F.7.67.69 getQualifierIsProtection

Description: Returns the isProtection boolean value of the given port qualifier element.

- Returns: `isProtection` of type **Boolean** - True if the isProtection qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.70 getQualifierIsRequest

Description: Returns the isRequest boolean value of the given port qualifier element.

- Returns: `isRequest` of type **Boolean** - True if the isRequest qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.71 getQualifierIsReset

Description: Returns the isReset boolean value of the given port qualifier element.

- Returns: `isReset` of type **Boolean** - True if the isReset qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.72 getQualifierIsResetID

Description: Returns the handle to the isReset element defined on the given port qualifier element.

- Returns: `isResetID` of type **String** - Handle to the isReset element
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.73 getQualifierIsResponse

Description: Returns the isResponse boolean value of the given port qualifier element.

- Returns: `isResponse` of type **Boolean** - True if the isResponse qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.74 getQualifierIsUser

Description: Returns the isUser boolean value of the given port qualifier element.

- Returns: `isUser` of type **Boolean** - True if the isUser qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.75 getQualifierIsUserID

Description: Returns the handle to the isUser element defined on the given port qualifier element.

- Returns: `isUserID` of type **String** - Handle to the isUser element
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.76 getQualifierIsValid

Description: Returns the isValid boolean value of the given port qualifier element.

- Returns: `isValid` of type **Boolean** - True if the isValid qualifier is defined
- Input: `qualifierID` of type **String** - Handle to a qualifier element

F.7.67.77 getServiceTypeDefServiceTypeDefIDs

Description: Returns the handles to all the serviceTypeDefs defined on the given serviceTypeDef element.

- Returns: `serviceTypeDefIDs` of type **String** - List of handles to the serviceTypeDef elements
- Input: `serviceTypeDefID` of type **String** - Handle to a serviceTypeDef element

F.7.67.78 getServiceTypeDefTypeDefinitionIDs

Description: Returns the handles to all the typeDefinitions defined on the given serviceTypeDef element.

- Returns: `typeDefinitionIDs` of type **String** - List of handles to the typeDefinition elements
- Input: `serviceTypeDefID` of type **String** - Handle to a serviceTypeDef element

F.7.67.79 getServiceTypeDefTypeName

Description: Returns the typeName defined on the given serviceTypeDef element.

- Returns: `typeName` of type **String** - The service type
- Input: `serviceTypeDefID` of type **String** - Handle to a serviceTypeDef element

F.7.67.80 getServiceTypeDefTypeID

Description: Returns the handle to a typeName element.

- Returns: `value` of type **String** - The handle to a typeName element
- Input: `serviceTypeDefID` of type **String** - Handle to a serviceTypeDef element

F.7.67.81 getServiceTypeDefTypeParameterIDs

Description: Returns the handles to all the typeParameters defined on the given serviceTypeDef element.

- Returns: `typeParameterIDs` of type ***String*** - List of handles to the typeParameter elements
- Input: `serviceTypeDefID` of type ***String*** - Handle to a serviceTypeDef element

F.7.67.82 getServiceTypeDefTypeParametersIDs

Description: Returns the handles to all the typeParameters defined on the given serviceTypeDef element.

- Returns: `typeParameterIDs` of type ***String*** - List of handles to the typeParameter elements
- Input: `serviceTypeDefID` of type ***String*** - Handles to a serviceTypeDef element

F.7.67.83 getSignalTypeDefSignalType

Description: Returns the signalType defined on the given signalTypeDef element.

- Returns: `signalType` of type ***String*** - The signalType value. Can be one of the following: continuous-conservative, continuous-non-conservative, discrete or digital.
- Input: `signalTypeDefID` of type ***String*** - Handle to a signalTypeDef

F.7.67.84 getSignalTypeDefViewIDs

Description: Returns the handles to all the views defined on the given typeDef element.

- Returns: `viewIDs` of type ***String*** - List of handles to the view elements
- Input: `signalTypeDefID` of type ***String*** - Handle to a signalTypeDef element

F.7.67.85 getSignalTypeDefViewRefsIDs

Description: Returns the handles to all the viewRefs defined on the given TypeDef element.

- Returns: `viewRefIDs` of type ***String*** - List of handles to the viewRef elements
- Input: `signalTypeDefID` of type ***String*** - Handle to a signalTypeDef element

F.7.67.86 getSignalTypeDefViewRefs

Description: Returns all the viewRefs defined on the given signalTypeDef element.

- Returns: `viewRefs` of type ***String*** - List of the referenced views
- Input: `signalTypeDefID` of type ***String*** - Handle to a signalTypeDef element

F.7.67.87 getStrucPortTypeDefViewRefsIDs

Description: Returns the handles to all the viewRefs defined on the given structPortTypeDef element.

- Returns: `viewRefIDs` of type ***String*** - List of handles to viewRef elements
- Input: `structPortTypeDefID` of type ***String*** - Handle to a structPortTypeDef element

F.7.67.88 getStructPortTypeDefRole

Description: Returns the role defined on the given structPortTypeDef element.

- Returns: `role` of type ***String*** - The structPort role
- Input: `structPortTypeDefID` of type ***String*** - Handle to a structPortTypeDef element

F.7.67.89 getStructPortTypeDefTypeDefinitionIDs

Description: Returns the handles to all the typeDefinitions defined on the given structPortTypeDef element.

- Returns: `typeDefinitionIDs` of type ***String*** - List of handles to the typeDefinition elements
- Input: `structPortTypeDefID` of type ***String*** - Handle to a structPortTypeDef element

F.7.67.90 getStructPortTypeDefTypeDefinitions

Description: Returns the typeDefinitions defined on the given structPortTypeDef element.

- Returns: `typeDefinitions` of type ***String*** - List of typeDefinitions
- Input: `structPortTypeDefID` of type ***String*** - Handle to a structPortTypeDef element

F.7.67.91 getStructPortTypeDefTypeName

Description: Returns the typeName defined on the given structPortTypeDef element.

- Returns: `typeName` of type ***String*** - The type name
- Input: `structPortTypeDefID` of type ***String*** - Handle to a structPortTypeDef element

F.7.67.92 getStructPortTypeDefTypeNameID

Description: Returns the handle to the typeName defined on the given structPortTypeDef element.

- Returns: `typeNameID` of type ***String*** - Handle to the typeName element
- Input: `structPortTypeDefID` of type ***String*** - Handle to a structPortTypeDef element

F.7.67.93 getStructPortTypeDefTypeParameterIDs

Description: Returns the handles to all the typeParameters defined on the given structPortTypeDef element.

- Returns: `typeParameterIDs` of type ***String*** - List of handles to the typeParameter elements
- Input: `structPortTypeDefID` of type ***String*** - Handle to a structPortTypeDef element

F.7.67.94 getStructPortTypeDefViewIDs

Description: Returns the handles to all the views defined on the given structPortTypeDef element.

- Returns: `viewIDs` of type ***String*** - List of handles to the view elements
- Input: `structPortTypeDefID` of type ***String*** - Handle to a structPortTypeDef element

F.7.67.95 getSubPortPartSelectID

Description: Returns the handles to all the partSelect defined on the given subPort element.

- Returns: `partSelectID` of type ***String*** - Handle to the partSelect element defined on the subPort element
- Input: `subPortID` of type ***String*** - Handle to a subPort

F.7.67.96 getSubPortReferencePartSelectID

Description: Returns the handle to the partSelect defined on the given subPortReference element.

- Returns: `partSelectID` of type ***String*** - Handle to the partSelect element
- Input: `subPortReferenceID` of type ***String*** - Handle to a subPortReference element

F.7.67.97 getSubPortReferenceSubPortRefByName

Description: Returns the subPortRef defined on the given subPortReference element.

- Returns: `subPortRef` of type ***String*** - The referenced subPort name
- Input: `subPortReferenceID` of type ***String*** - Handle to a subPortReference element

F.7.67.98 getTransTypeDefServiceTypeDefIDs

Description: Returns the handles to all the serviceTypeDefs defined on the given transTypeDef element.

- Returns: `serviceTypeDefIDs` of type ***String*** - List of handles to the serviceTypeDef elements
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef

F.7.67.99 getTransTypeDefTypeDefinitionIDs

Description: Returns the handles to all the typeDefinitions defined on the given transTypeDef element.

- Returns: `typeDefinitionIDs` of type ***String*** - List of handles to the typeDefinition elements
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element

F.7.67.100 getTransTypeDefTypeDefinitions

Description: Returns the typeDefinitions defined on the given transTypeDef element.

- Returns: `typeDefinitions` of type ***String*** - List of typeDefinitions
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element

F.7.67.101 getTransTypeDefTypeName

Description: Returns the typeName defined on the given transTypeDef element.

- Returns: `typeName` of type ***String*** - The type name
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element

F.7.67.102 getTransTypeDefTypeNameID

Description: Returns the handle to the typeName defined on the given transTypeDef element.

- Returns: `typeNameID` of type ***String*** - Handle to the typeName element
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element

F.7.67.103 getTransTypeDefTypeParameterIDs

Description: Returns the handles to all the typeParameters defined on the given transTypeDef element.

- Returns: `typeParameterIDs` of type ***String*** - List of handles to the typeParameter elements
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef

F.7.67.104 getTransTypeDefTypeParametersIDs

Description: Returns the handles to all the typeParameters defined on the given transTypeDef element.

- Returns: `typeParameterIDs` of type ***String*** - List of handles to the typeParameter elements
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element

F.7.67.105 getTransTypeDefViewRefByID

Description: Returns the view defined on the given viewRef element.

- Returns: `viewID` of type ***String*** - Handles to the referenced view element
- Input: `transTypeDefID` of type ***String*** - Handle to the transTypeDef element
- Input: `viewRef` of type ***String*** - Handle to the viewRef element

F.7.67.106 getTransTypeDefViewRefIDs

Description: Returns the handles to all the viewRefs defined on the given transTypeDef element.

- Returns: `viewRefIDs` of type ***String*** - List of handles to the viewRef elements
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element

F.7.67.107 getWireTypeDefTypeDefinitionIDs

Description: Returns the handles to all the typeDefinitions defined on the given wireTypeDef element.

- Returns: `typeDefinitionIDs` of type ***String*** - List of handles to the typeDefinition elements
- Input: `wireTypeDefID` of type ***String*** - Handle to a wireTypeDef element

F.7.67.108 getWireTypeDefTypeDefinitions

Description: Returns the typeDefinitions defined on the given wireTypeDef element.

- Returns: `typeDefinitions` of type ***String*** - List of typeDefinitions
- Input: `wireTypeDefID` of type ***String*** - Handle to a wireTypeDef element

F.7.67.109 getWireTypeDefTypeName

Description: Returns the typeName defined on the given wireTypeDef element.

- Returns: `typeName` of type ***String*** - The type name
- Input: `wireTypeDefID` of type ***String*** - Handle to a wireTypeDef element

F.7.67.110 getWireTypeDefTypeNameID

Description: Returns the handle to the typeName defined on the given wireTypeDef element.

- Returns: `typeNameID` of type ***String*** - Handle to the typeName element
- Input: `wireTypeDefID` of type ***String*** - Handle to a wireTypeDef element

F.7.67.111 getWireTypeDefViewIDs

Description: Returns the handles to all the views defined on the given wireTypeDef element.

- Returns: `viewIDs` of type ***String*** - List of handles to the view elements
- Input: `wireTypeDefID` of type ***String*** - Handle to a wireTypeDef element

F.7.67.112 getWireTypeDefViewRefIDs

Description: Returns the handles to all the viewRefs defined on the given wireTypeDef element.

- Returns: `viewRefIDs` of type ***String*** - List of handles to the viewRef elements
- Input: `wireTypeDefID` of type ***String*** - Handle to a wireTypeDef element

F.7.67.113 getWireTypeDefViewRefs

Description: Returns all the viewRefs defined on the given wireTypeDef element.

- Returns: `viewRefs` of type ***String*** - List of the referenced views
- Input: `wireTypeDefID` of type ***String*** - Handle to a wireTypeDef element

F.7.68 Port (EXTENDED)

F.7.68.1 addDomainTypeDefTypeDefinition

Description: Sets the typeDefinitions with the given definitions for the given domainTypeDef element.

- Returns: `typeDefinitionID` of type ***String*** - Handle of the new typeDefinition.
- Input: `domainTypeDefID` of type ***String*** - Handle to a domainTypeDef element
- Input: `typeDefinition` of type ***String*** - The type definition value

F.7.68.2 addDomainTypeDefViewRef

Description: Adds the given viewRef in the given domainTypeDef element.

- Returns: `viewRefID` of type ***String*** - Handle to the added viewRef
- Input: `domainTypeDefID` of type ***String*** - Handle to a domainTypeDef element
- Input: `viewRef` of type ***String*** - Name of the referenced view

F.7.68.3 addExternalPortReferenceSubPortReference

Description: Adds a reference to a subPort to the given externalPortReference of a structured port.

- Returns: `subPortReferenceID` of type ***String*** - The identifier of the added subPortReference
- Input: `externalPortReferenceID` of type ***String*** - Handle to an externalPortReference element
- Input: `subPortRef` of type ***String*** - Name of a subPort of the structured port

F.7.68.4 addFieldMapIndex

Description: Adds an index to the indices of the fieldMap element.

- Returns: `indexID` of type ***String*** - an identifier to the added index
- Input: `fieldMapID` of type ***String*** - Handle to a fieldMap element
- Input: `value` of type ***String*** - Index value expression

F.7.68.5 addFieldMapModeRef

Description: Adds a modeRef to a fieldMap element.

- Returns: `modeRefID` of type ***String*** - Handle to a modeRef element
- Input: `fieldMapID` of type ***String*** - Handle to a fieldMap element
- Input: `modeRef` of type ***String*** - the modeRef value to be added

F.7.68.6 addFieldMapSubPortReference

Description: Adds a subPortReference to a fieldMap element.

- Returns: `subPortReferenceID` of type ***String*** - the subPortReference identifier

- Input: fieldMapID of type **String** - Handle to a fieldMap element
- Input: subPortRef of type **String** - the subPort to be added

F.7.68.7 addInternalPortReferenceSubPortReference

Description: Adds a subPortReference to a subPort to the given internalPortReference of a structured port.

- Returns: subPortReferenceID of type **String** - The identifier of the added subPortReference
- Input: internalPortReferenceID of type **String** - Handle to an internalPortReference
- Input: subPortRef of type **String** - Name of a subPort of the structured port

F.7.68.8 addPortClockDriver

Description: Adds clockDriver with the given clock period, pulse offset, pulse value, and pulse duration to the given port element.

- Returns: clockDriverID of type **String** - Handle to a new clockDriver
- Input: portID of type **String** - Handle to a port element
- Input: period of type **Float** - Clock period
- Input: offset of type **Float** - Clock pulse offset
- Input: value of type **Long** - Clock pulse value
- Input: duration of type **Float** - Clock pulse duration

F.7.68.9 addPortClockDriverExpresion

Description: Adds clockDriver with the given clock period, pulse offset, pulse value, and pulse duration to the given port element.

- Returns: clockDriverID of type **String** - Handle to a new clockDriver
- Input: portID of type **String** - Handle to a port element
- Input: periodExpression of type **String** - Clock period
- Input: offsetExpression of type **String** - Clock pulse offset
- Input: valueExpression of type **String** - Clock pulse value
- Input: durationExpression of type **String** - Clock pulse duration

F.7.68.10 addPortDefaultDriver

Description: Adds defaultDriver with the given value to the given port element.

- Returns: driverID of type **String** - Handle to a new driver
- Input: portID of type **String** - Handle to a port element
- Input: value of type **Long** - Default driver value

F.7.68.11 addPortDefaultDriverExpression

Description: Adds defaultDriver with the given value to the given port element.

- Returns: driverID of type **String** - Handle to a new driver
- Input: portID of type **String** - Handle to a port element
- Input: valueExpression of type **String** - Default driver value

F.7.68.12 addPortDomainTypeDef

Description: Adds domainTypeDef for the given port wire element.

- Returns: domainTypeDefID of type **String** - Handle to the added domainTypeDef
- Input: wireID of type **String** - Handle to a port wire element

F.7.68.13 addPortFieldMap

Description: Adds a new fieldMap.

- Returns: portmapID of type **String** - Handle to the added fieldMap
- Input: portID of type **String** - Handle to a port element
- Input: memoryMapRef of type **String** - MemoryMap name (non null)
- Input: addressBlockRef of type **String** - AddressBlock name (non null)
- Input: registerRef of type **String** - Register name (non null)
- Input: fieldRef of type **String** - Field name (non null)

F.7.68.14 addPortSignalTypeDef

Description: Adds domainTypeDef for the given port wire element.

- Returns: signalTypeDefID of type **String** - Handle to the added signalTypeDef
- Input: wireID of type **String** - Handle to a port wire element
- Input: signalType of type **String** - The type of the signal. Can be one of continuous-conservative or continuous-non-conservative or discrete or digital

F.7.68.15 addPortSingleShotDriver

Description: Adds singleShotDriver with the given offset, value, and duration to the given port element.

- Returns: singleShotDriverID of type **String** - Handle to a new singleShotDriver
- Input: portID of type **String** - Handle to a port element
- Input: offset of type **Float** - SingleShot offset
- Input: value of type **Long** - SingleShot value
- Input: duration of type **Float** - SingleShot duration

F.7.68.16 addPortSingleShotDriverExpression

Description: Adds singleShotDriver with the given offset, value, and duration to the given port element.

- Returns: singleShotDriverID of type **String** - Handle to a new singleShotDriver
- Input: portID of type **String** - Handle to a port element
- Input: offsetExpression of type **String** - The singleShot offset
- Input: valueExpression of type **String** - The singleShot value
- Input: durationExpression of type **String** - The singleShot duration

F.7.68.17 addPortStructuredStructPortTypeDef

Description: Adds a structPortTypeDef to the containing element.

- Returns: structPortTypeDefID of type **String** - Handle to the added structPortTypeDef
- Input: structuredID of type **String** - Handle to a port structured element
- Input: typeName of type **String** - Structured port definition type name

F.7.68.18 addPortStructuredSubStructuredPort

Description: Adds a subPort with structured to a port element.

- Returns: subPortID of type **String** - the ID of the created subPort
- Input: portID of type **String** - Handle to a port
- Input: subPortName of type **String** - name on the subPort
- Input: vectorLeft of type **String** - The vector left expression
- Input: vectorRight of type **String** - The vector right expression
- Input: subSubPortName of type **String** - The name of the subPort inside the subPort
- Input: structPortTypeDefTypeName of type **String** - The typeName of the structPortTypeDef

F.7.68.19 addPortStructuredSubWirePort

Description: Adds a subPort with wire to a port structured element.

- Returns: subPortID of type **String** - The ID of the created subPort
- Input: structuredID of type **String** - Handle to structured
- Input: subPortName of type **String** - The name on the subPort
- Input: direction of type **String** - The port direction

F.7.68.20 addPortStructuredVector

Description: Adds a vector from a structured port.

- Returns: vectorID of type **String** - The vector identifier
- Input: structuredID of type **String** - Handle to a port structured element
- Input: left of type **String** - The left value
- Input: right of type **String** - The right value

F.7.68.21 addPortTransactionalTransTypeDef

Description: Adds transTypeDef for the given port element.

- Returns: transTypeDefID of type **String** - Handle to a new transTypeDef
- Input: transactionalID of type **String** - Handle to a port transactional element

F.7.68.22 addPortWireConstraintSet

Description: Adds constraintSet to the given wire element of a port.

- Returns: constraintSetID of type **String** - Handle to a new constraintSet
- Input: wireID of type **String** - Handle to a wire element

F.7.68.23 addPortWireDriver

Description: Adds driver without a value to the given port element.

- Returns: driverID of type **String** - Handle to a new driver
- Input: portWireID of type **String** - Handle to a port wire element
- Input: defaultValue of type **String** - The defaultValue expression

F.7.68.24 addPortWireTypeDef

Description: Adds wireTypeDef for the given port element.

- Returns: wireTypeDefID of type **String** - Handle to a new wireTypeDef
- Input: wireTypeID of type **String** - Handle to a port wire

F.7.68.25 addServiceTypeDefServiceTypeDef

Description: Adds a serviceTypeDef to an existing serviceTypeDef on his typeParameters.

- Returns: serviceTypeDefID of type **String** - Handle to the added serviceTypeDef
- Input: serviceTypeDefID of type **String** - Handle to a serviceTypeDef element
- Input: typeName of type **String** - Service type name

F.7.68.26 addServiceTypeDefTypeDefinition

Description: Adds a TypeDefinition to the containing element.

- Returns: typeDefinitionID of type **String** - Handle to the added typeDefinition
- Input: serviceTypeDefID of type **String** - Handle to a serviceTypeDef element
- Input: typeDefinition of type **String** - Name of the file where the type is defined

F.7.68.27 addServiceTypeDefTypeParameter

Description: Adds a TypeParameter to the containing element.

- Returns: typeParameterID of type **String** - Handle to the added typeParameter
- Input: serviceTypeDefID of type **String** - Handle to a serviceTypeDef element
- Input: name of type **String** - Name of the parameter
- Input: value of type **String** - Value of the parameter

F.7.68.28 addSignalTypeDefViewRef

Description: Adds the given viewRef in the given signalTypeDef element.

- Returns: viewRefID of type **String** - Handle to the added viewRef
- Input: signalTypeDefID of type **String** - Handle to a signalTypeDef element
- Input: viewRef of type **String** - Name of the referenced view

F.7.68.29 addStructPortTypeDefTypeDefinition

Description: Adds typeDefinition to the containing element.

- Returns: typeDefinitionID of type **String** - Handle to the added typeDefinition
- Input: structPortTypeDefID of type **String** - Handle to a structPortTypeDef element
- Input: value of type **String** - File name containing the definition of the structured port type

F.7.68.30 addStructPortTypeDefTypeParameter

Description: Adds typeParameter to the containing element.

- Returns: typeParameterID of type **String** - Handle to the added typeParameter
- Input: structPortTypeDefID of type **String** - Handle to a strucPortTypeDef
- Input: name of type **String** - Parameter name

- Input: value of type **String** - Parameter value

F.7.68.31 addStructPortTypeDefViewRef

Description: Adds the given viewRef in the given structPortTypeDef element.

- Returns: viewRefID of type **String** - Handle to the added viewRef
- Input: structPortTypeDefID of type **String** - Handle to a structPortTypeDef element
- Input: viewRef of type **String** - Name of the referenced view

F.7.68.32 addTransTypeDefServiceTypeDef

Description: Adds a serviceTypeDef to the containing element.

- Returns: serviceTypeDefID of type **String** - Handle to the added serviceTypeDef
- Input: transTypeDefID of type **String** - Handle to a transTypeDef element
- Input: typeName of type **String** - Type name

F.7.68.33 addTransTypeDefTypeDefinition

Description: Adds a typeDefinition to the given transactional port type definition.

- Returns: typeDefinitionID of type **String** - The identifier of the added typeDefinition
- Input: transTypeDefID of type **String** - Handle to transTypeDef element
- Input: value of type **String** - Value of the added typeDefinition

F.7.68.34 addTransTypeDefTypeParameter

Description: Adds a typeParameter to the containing element.

- Returns: typeParameterID of type **String** - Handle to the added typeParameter
- Input: transTypeDefID of type **String** - Handle to a transTypeDef element
- Input: name of type **String** - Parameter name
- Input: value of type **String** - Parameter value

F.7.68.35 addTransTypeDefViewRef

Description: Adds the given viewRef in the given transTypeDef element.

- Returns: viewRefID of type **String** - Handle to the added viewRef
- Input: transTypeDefID of type **String** - Handle to a transTypeDef element
- Input: viewRef of type **String** - Name of the referenced view

F.7.68.36 addWireTypeDefTypeDefinition

Description: Adds the typeDefinition with the given definition for the given wireTypeDef element.

- Returns: typeDefinitionID of type **String** - The typeDefinition identifier
- Input: wireTypeDefID of type **String** - Handle to a wireTypeDef element
- Input: typeDefinition of type **String** - The Type definition to add

F.7.68.37 addWireTypeDefViewRef

Description: Adds the given viewRef in the given wireTypeDef element.

- Returns: viewRefID of type **String** - Handle to the added viewRef

- Input: wireTypeDefID of type **String** - Handle to a wireTypeDef element
- Input: viewRef of type **String** - Name of the referenced view

F.7.68.38 removeAbstractionDefPortPacket

Description: Removes the given packet from its containing logical port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: packetID of type **String** - Handle to a portPacketType element

F.7.68.39 removeAccessPortAccessType

Description: Removes portAccessType from the given access element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessID of type **String** - Handle to an access element

F.7.68.40 removeAllLogicalDirectionsAllowedAttribute

Description: Removes allLogicalDirectionsAllowed from the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: wireID of type **String** - Handle to a wire element

F.7.68.41 removeAllLogicalInitiativesAllowedAttribute

Description: Removes allLogicalInitiativesAllowed from the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a transactional element

F.7.68.42 removeDomainTypeDefTypeDefinition

Description: Removes the given typeDefinition from the domainTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: typeDefinitionID of type **String** - Handle of type definition

F.7.68.43 removeDomainTypeDefTypeName

Description: Removes the typeName with the given value for the given domainTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element

F.7.68.44 removeDomainTypeDefViewRef

Description: Removes the given viewRef from its containing domainTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewRefID of type **String** - Handle to a viewRef element

F.7.68.45 removeExactAttribute

Description: Removes the exact attribute of the transactional port type definition type name.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: `transTypeDefID` of type ***String*** - Handle to a `transTypeDef` element

F.7.68.46 removeFieldMapModeRef

Description: Removes the given modeRef from its containing fieldMap element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `modeRefID` of type ***String*** - the modeRef value to be removed

F.7.68.47 removeFieldMapPartSelect

Description: Removes a partSelect on the given fieldMap element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fieldMapID` of type ***String*** - Handle to a `portMap` fieldMap element

F.7.68.48 removeFieldMapSubPortReference

Description: Removes the given subPortReference from its containing fieldMap element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `subPortRefID` of type ***String*** - the subPort to be removed

F.7.68.49 removePayloadExtension

Description: Removes payload extension from the given port element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `payloadID` of type ***String*** - Handle to a protocol payload element

F.7.68.50 removePortAccess

Description: Removes port access from the given port element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.51 removePortClockDriver

Description: Removes the given clockDriver element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `clockDriverID` of type ***String*** - Handle to a `clockDriver` element

F.7.68.52 removePortDefaultDriver

Description: Removes the given defaultDriver element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.68.53 removePortDomainTypeDef

Description: Removes the given domainTypeDef from its containing port element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `domainTypeDefID` of type ***String*** - Handle to a `domainTypeDef` element

F.7.68.54 removePortFieldMap

Description: Removes the given fieldMap from its containing port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldMapID of type **String** - Handle to a fieldMap

F.7.68.55 removePortSignalTypeDef

Description: Removes the given signalTypeDef from its containing port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: signalTypeDefID of type **String** - Handle to a signalTypeDef element

F.7.68.56 removePortSingleShotDriver

Description: Removes the given singleShot driver element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: singleShotDriverID of type **String** - Handle to a singleShot driver element

F.7.68.57 removePortStructuredStructPortTypeDef

Description: Removes the given structPortTypeDef from its containing port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: structPortTypeDefID of type **String** - Handle to a strucPortTypeDef

F.7.68.58 removePortStructuredSubPort

Description: Removes the given subPort form its containing structured port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subPortID of type **String** - Handle to a subPort element

F.7.68.59 removePortStructuredVector

Description: Removes the given vector from its containing structured port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: vectorID of type **String** - Handle to a vector element

F.7.68.60 removePortTransactionalBusWidth

Description: Removes busWidth from the given port transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element

F.7.68.61 removePortTransactionalKind

Description: Removes kind from the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element

F.7.68.62 removePortTransactionalMaxConnections

Description: Removes maxConnections from the given port transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element

F.7.68.63 removePortTransactionalMinConnections

Description: Sets minConnections from the given port transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element

F.7.68.64 removePortTransactionalPowerConstraint

Description: Removes the given power constraint from its containing transactional port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerConstraintID of type **String** - Handle to a power constraint

F.7.68.65 removePortTransactionalProtocol

Description: Removes protocol from transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to the identifier of a port transactional

F.7.68.66 removePortTransactionalQualifier

Description: Removes the given qualifier from its contained component port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - Handle to qualifier

F.7.68.67 removePortTransactionalTransTypeDef

Description: Removes the given transTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transTypeDefID of type **String** - Handle to a transTypeDef element

F.7.68.68 removePortWireConstraintSet

Description: Removes the given constraintSet element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: constraintSetID of type **String** - Handle to a constraintSet element

F.7.68.69 removePortWireDriver

Description: Removes driver with the given driverID on wire port element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: driverID of type **String** - Handle to a driver element

F.7.68.70 removePortWirePowerConstraint

Description: Removes the given power constraint from its containing wire port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerConstraintID of type **String** - Handle to a power constraint

F.7.68.71 removePortWireQualifier

Description: Removes the given qualifier from its contained component port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - Handle to qualifier

F.7.68.72 removePortWireTypeDef

Description: Removes the given wireTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: wireTypeDefID of type **String** - Handle to a wireTypeDef element

F.7.68.73 removeProtocolPayload

Description: Removes custom attribute of protocolType from the given abstractionDefPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: protocolID of type **String** - Handle to protocol element

F.7.68.74 removeQualifierIsAddress

Description: Removes the isAddress qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.75 removeQualifierIsClock

Description: Removes the isClock qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.76 removeQualifierIsClockEn

Description: Removes the isClockEn qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.77 removeQualifierIsData

Description: Removes the isData qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.78 removeQualifierIsFlowControl

Description: Removes the isFlowControl qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.79 removeQualifierIsInterrupt

Description: Removes the isInterrupt qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.80 removeQualifierIsOpcode

Description: Removes the isOpCode qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.81 removeQualifierIsPowerEn

Description: Removes the isPowerEn qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.82 removeQualifierIsProtection

Description: Removes the isProtection qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.83 removeQualifierIsRequest

Description: Removes the isRequest qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.84 removeQualifierIsReset

Description: Removes the isReset qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.85 removeQualifierIsResponse

Description: Removes the isResponse qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.86 removeQualifierIsUser

Description: Removes the isUser qualifier of for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.87 removeQualifierIsValid

Description: Removes the isValid qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

F.7.68.88 removeServiceTypeDefServiceTypeDef

Description: Removes the given serviceTypeDef from its containing transactional port serviceTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: serviceTypeDefID of type **String** - Handle to a serviceTypeDef element

F.7.68.89 removeServiceTypeDefTypeDefinition

Description: Removes the given typeDefinition from its containing transactional port serviceTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: typeDefinitionID of type **String** - Handle to a typeDefinition

F.7.68.90 removeServiceTypeDefTypeParameter

Description: Removes the given typeParameter from its containing transactional port serviceTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: typeParameterID of type **String** - Handle to a typeParameter

F.7.68.91 removeSignalTypeDefViewRef

Description: Removes the given viewRef from its containing signalTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewRefID of type **String** - Handle to a viewRef element

F.7.68.92 removeStructPortTypeDefRole

Description: Removes the role associated with the given structured port typeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: structPortTypeDefID of type **String** - Handle to a structPortTypeDef element

F.7.68.93 removeStructPortTypeDefTypeDefinition

Description: Removes the given typeDefinition from its containing strucPortTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: typeDefinitionID of type **String** - Handle to a typeDefinition element

F.7.68.94 removeStructPortTypeDefTypeParameter

Description: Removes the given typeParameter from its containing structPortTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: typeParameterID of type **String** - Handle to a typeParameter element

F.7.68.95 removeStructPortTypeDefViewRef

Description: Removes the given viewRef from its containing structPortTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewRefID of type **String** - Handle to a viewRef element

F.7.68.96 removeSubPortMapPartSelect

Description: Removes a Part Select on a subPortMap element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subPortMapID of type **String** - Handle to subPortMap element

F.7.68.97 removeSubPortReferencePartSelect

Description: Removes a partSelect on the given subPortReference element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subPortReferenceID of type **String** - Handle to a portMap subPortReference element

F.7.68.98 removeTransTypeDefServiceTypeDef

Description: Removes the given serviceTypeDef from its containing transactional port type definition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: serviceTypeDefID of type **String** - Handle to a serviceTypeDef

F.7.68.99 removeTransTypeDefTypeDefinition

Description: Removes the given typeDefinition from its containing the transactional port type definition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: typeDefinitionID of type **String** - Handle to a typeDefinition element

F.7.68.100 removeTransTypeDefTypeName

Description: Removes the name of the transactional port type definition.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transTypeDefID of type **String** - Handle to a transTypeDef element

F.7.68.101 removeTransTypeDefTypeParameter

Description: Removes the given typeParameter from its containing transactional port type definition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: typeParameterID of type **String** - Handle to a typeParameter

F.7.68.102 removeTransTypeDefViewRef

Description: Removes the given viewRef from its containing transTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewRefID of type **String** - Handle to a viewRef element

F.7.68.103 removeWireTypeDefTypeDefinition

Description: Removes the typeDefinition from the wireTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: typeDefinitionID of type **String** - Handle to type definition to remove

F.7.68.104 removeWireTypeDefTypeName

Description: Removes the typeName with the given value for the given wireTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: wireTypeDefID of type **String** - Handle to a wireTypeDef element

F.7.68.105 removeWireTypeDefViewRef

Description: Removes the given viewRef from its containing wireTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewRefID of type **String** - Handle to a viewRef element

F.7.68.106 setAbstractionDefBusType

Description: Sets the busType vlnv for the given abstraction definition object.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionDefinitionID of type **String** - Handle to the abstractionDefinition
- Input: vlnv of type **String[]** - VLN of the busDefinition

F.7.68.107 setAccessPortAccessType

Description: Sets portAccessType with the given value for the given access element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: accessID of type **String** - Handle to an access element
- Input: accessType of type **String** - Port portAccessType

F.7.68.108 setExternalPortReferencePortReference

Description: Sets the portRef of the externalPort.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: externalPortReferenceID of type **String** - Handle to an externalPortReference element
- Input: value of type **String** - Name of the referenced port

F.7.68.109 setInternalPortReferenceComponentInstanceReference

Description: Sets the componentInstanceRef for the internal port reference.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: internalPortReferenceID of type **String** - Handle to an internalPortReference element
- Input: value of type **String** - Name of the referenced componentInstance

F.7.68.110 setInternalPortReferencePortReference

Description: Sets the portRef for the internal port reference.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: internalPortReferenceID of type **String** - Handle to an internalPortReference element
- Input: value of type **String** - Name of the referenced port

F.7.68.111 setDomainTypeDefTypeName

Description: Sets the typeName with the given value for the given domainTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element
- Input: typeName of type **String** - Value of the typeName element

F.7.68.112 setFieldMapAddressBlockRef

Description: Sets the addressBlockRef of the given fieldMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldMapID of type **String** - Handle to a fieldMap element
- Input: value of type **String** - AddressBlock name

F.7.68.113 setFieldMapFieldRef

Description: Sets the fieldRef of the given fieldMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldMapID of type **String** - Handle to a fieldMap element
- Input: value of type **String** - Field name

F.7.68.114 setFieldMapFieldSlice

Description: Sets the fieldSlice references on the given fieldMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldMapID of type **String** - Handle to a fieldMap element on a port
- Input: memoryMapRef of type **String** - Name of the referenced memoryMap
- Input: addressBlockRef of type **String** - Name of the referenced addressBlock
- Input: registerRef of type **String** - Name of the referenced register
- Input: fieldRef of type **String** - Name of the referenced field

F.7.68.115 setFieldMapMemoryMapRef

Description: Sets the memoryMapRef of the given fieldMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: `fieldMapID` of type ***String*** - Handle to a fieldMap element
- Input: `value` of type ***String*** - MemoryMap name

F.7.68.116 setFieldMapPartSelect

Description: Sets a partSelect on the given fieldMap element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fieldMapID` of type ***String*** - Handle to a portMap fieldMap element
- Input: `range` of type ***String[]*** - Create the range on the partSelect with “left” for range[0] and “right” for range[1]. Set to null if you only want indices.
- Input: `indices` of type ***String[]*** - Handle to values of type String. Set all the index on the partSelect

F.7.68.117 setFieldMapRegisterRef

Description: Sets the registerRef of the given fieldMap element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fieldMapID` of type ***String*** - Handle to a fieldMap element
- Input: `value` of type ***String*** - Register name

F.7.68.118 setPayloadExtension

Description: Sets the name of the extension for the given payload.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `payloadID` of type ***String*** - Handle to a payload element
- Input: `extension` of type ***String*** - Name of the extension

F.7.68.119 setPayloadType

Description: Sets the type on a payload element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `payloadID` of type ***String*** - Handle to a payload element
- Input: `type` of type ***String*** - The payload type to set

F.7.68.120 setPortAccess

Description: Sets port access with the given value for the given port element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.121 setPortStructured

Description: Sets the structured element in the given port.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to the identifier of a port element
- Input: `subPortName` of type ***String*** - The name of the subPort
- Input: `structPortTypeDefTypeName` of type ***String*** - The typeName of the structPortTypeDef

F.7.68.122 setPortStructuredInterface

Description: Sets the interface on the given structured element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: structuredID of type **String** - Handle to a port structured
- Input: interfaceValue of type **Boolean** - The interface to set

F.7.68.123 setPortStructuredStruct

Description: Sets the struct element on the given structured element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: structuredID of type **String** - Handle to a port structured
- Input: direction of type **String** - The Struct direction to set

F.7.68.124 setPortStructuredUnion

Description: Sets the union element on the given structured element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: structuredID of type **String** - Handle to a port structured
- Input: direction of type **String** - The Union direction to set

F.7.68.125 setPortTransactional

Description: Sets the transactional element in the given port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portID of type **String** - Handle to the identifier of a port element
- Input: initiative of type **String** - The direction of the wire port. Can be one of the following: requires, provides, both, or phantom

F.7.68.126 setPortTransactionalBusWidth

Description: Sets busWidth with the given value for the given port transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element
- Input: widthExpression of type **String** - Port busWidth

F.7.68.127 setPortTransactionalInitiative

Description: Sets initiative with the given value for the given port transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element
- Input: initiative of type **String** - Transactional port initiative

F.7.68.128 setPortTransactionalKind

Description: Sets kind with the given value for the given port transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element

- Input: kind of type **String** - Port kind

F.7.68.129 setPortTransactionalMaxConnections

Description: Sets maxConnections with the given value for the given port transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element
- Input: value of type **String** - Port maxConnections

F.7.68.130 setPortTransactionalMinConnections

Description: Sets minConnections with the given value for the given port transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to a port transactional element
- Input: value of type **String** - Port minConnections

F.7.68.131 setPortTransactionalProtocol

Description: Sets the protocol element on transactional element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to the identifier of a port transactional
- Input: type of type **String** - protocol type. Can be one of the following: tlm or custom

F.7.68.132 setPortTransactionalQualifier

Description: Sets the qualifier on the given component port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transactionalID of type **String** - Handle to port transactional

F.7.68.133 setPortWire

Description: Sets the wire element on the given port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portID of type **String** - Handle to the identifier of a port element
- Input: direction of type **String** - The direction of the wire port. Can be one of the following: in, out, inout, or phantom

F.7.68.134 setPortWireDirection

Description: Sets direction with the given value for the given port wire element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: wireID of type **String** - Handle to a port wire element
- Input: direction of type **String** - Port direction

F.7.68.135 setPortWireQualifier

Description: Sets the qualifier on the given component port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: wireID of type **String** - Handle to port wire

F.7.68.136 setProtocolPayload

Description: Sets protocolType from the given protocol element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: protocolID of type **String** - Handle to protocol element
- Input: type of type **String** - type value. Can be one of the following: generic or specific

F.7.68.137 setProtocolProtocolType

Description: Sets custom attribute of protocolType from the given protocol element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: protocolID of type **String** - Handle to protocol element
- Input: protocolType of type **String** - protocol type value. Can be one of the following: tlm or custom

F.7.68.138 setQualifierIsAddress

Description: Sets the isAddress qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier
- Input: value of type **Boolean** - True if the port is an address (null to unset)

F.7.68.139 setQualifierIsClock

Description: Sets the isClock qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier
- Input: value of type **Boolean** - True if the port is a clock (null to unset)

F.7.68.140 setQualifierIsClockEn

Description: Sets the isClockEn qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier
- Input: value of type **Boolean** - True if the port is a clock enable (null to unset)

F.7.68.141 setQualifierIsData

Description: Sets the isData qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier
- Input: value of type **Boolean** - True if the port is a data (null to unset)

F.7.68.142 setQualifierIsFlowControl

Description: Sets the isFlowControl qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier

- Input: value of type **Boolean** - True if the port is a flow control (null to unset)

F.7.68.143 setQualifierIsInterrupt

Description: Sets the isInterrupt qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier
- Input: value of type **Boolean** - True if the port is an interrupt (null to unset)

F.7.68.144 setQualifierIsOpcode

Description: Sets the isOpCode qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - the qualifier identifier
- Input: value of type **Boolean** - True if the port is an opCode (null to unset)

F.7.68.145 setQualifierIsPowerEn

Description: Sets the isPowerEn qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - The qualifier identifier
- Input: value of type **Boolean** - True if the port is a power enable (null to unset)

F.7.68.146 setQualifierIsProtection

Description: Sets the isProtection qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - The qualifier identifier
- Input: value of type **Boolean** - True if the port is a protection (null to unset)

F.7.68.147 setQualifierIsRequest

Description: Sets the isRequest qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - The qualifier identifier
- Input: value of type **Boolean** - True if the port is a request (null to unset)

F.7.68.148 setQualifierIsReset

Description: Sets the isReset qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - The qualifier identifier
- Input: value of type **Boolean** - True if the port is a reset (null to unset)

F.7.68.149 setQualifierIsResponse

Description: Sets the isResponse qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - The qualifier identifier

- Input: value of type **Boolean** - True if the port is a response (null to unset)

F.7.68.150 setQualifierIsUser

Description: Sets the isUser qualifier of for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - The qualifier identifier
- Input: value of type **Boolean** - True if the port is an user defined qualifier (null to unset)

F.7.68.151 setQualifierIsValid

Description: Sets the isValid qualifier for the given port element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: qualifierID of type **String** - The qualifier identifier
- Input: value of type **Boolean** - True if the port is valid (null to unset)

F.7.68.152 setServiceTypeDefTypeName

Description: Sets the name of the transactional port service type definition.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: serviceTypeDefID of type **String** - Handle to a serviceTypeDef element
- Input: typeName of type **String** - Name of the service type

F.7.68.153 setSignalTypeDefSignalType

Description: Sets the signal type defined to the given signalTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: signalTypeDefID of type **String** - Handle to the identifier of a signalTypeDef
- Input: signalType of type **String** - Name of the signalType. Can be one of the following: continuous-conservative, continuous-non-conservative, or discrete or digital.

F.7.68.154 setStructPortTypeDefRole

Description: Sets the role associated with the given structured port typeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: strucPortTypeDefID of type **String** - Handle to a structport typeDef element
- Input: role of type **String** - Role of this port

F.7.68.155 setStructPortTypeDefTypeName

Description: Sets the type name of the structured port.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: strucPortTypeDefID of type **String** - Handle to a strucPortTypeDef
- Input: typename of type **String** - Type name

F.7.68.156 setSubPortReferencePartSelect

Description: Set a partSelect on the given subPortReference element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: subPortReferenceID of type **String** - Handle to a portMap subPortReference element
- Input: range of type **String[]** - Create the range on the partSelect with “left” for range[0] and “right” for range[1]. Set to null if you only want indices.
- Input: indices of type **String[]** - Handle to values of type String. Set all the index on the partSelect

F.7.68.157 setTransTypeDefTypeName

Description: Sets the name of the transactional port type definition.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: transTypeDefID of type **String** - Handle to a transTypeDef element
- Input: typeName of type **String** - Port type name. For example, in SystemC, it could be: sc_port, or sc_export, or my_tlm_port.

F.7.68.158 setWireTypeDefTypeName

Description: Sets the typeName with the given value for the given wireTypeDef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: wireTypeDefID of type **String** - Handle to a wireTypeDef element
- Input: typeName of type **String** - Value of the typeName element

F.7.69 Port map (BASE)

F.7.69.1 getLogicalPortRange

Description: Returns the range left and right (resolved) value from the given logicalPort element.

- Returns: rangeValues of type **Long** - Array of two range values: left and right
- Input: logicalPortID of type **String** - Handle to a logicalPort element

F.7.69.2 getLogicalPortRangeExpression

Description: Returns the range left and right expressions defined on the given logicalPort element.

- Returns: rangeExpressions of type **String** - Array of two range expressions: left and right
- Input: logicalPortID of type **String** - Handle to a logicalPort element

F.7.69.3 getLogicalPortRangeLeftID

Description: Returns the handle to the left element defined on a logicalPort element.

- Returns: leftID of type **String** - Handle to the left element of the range
- Input: logicalPortID of type **String** - Handle to a logicalPort element

F.7.69.4 getLogicalPortRangeRightID

Description: Returns the handle to the right element defined on a logicalPort element.

- Returns: rightID of type **String** - Handle to the right element of the range
- Input: logicalPortID of type **String** - Handle to a logicalPort element

F.7.69.5 getPhysicalPortPartSelectID

Description: Returns the handle to the partSelect defined on the given physicalPort element.

- Returns: `partSelectID` of type ***String*** - Handle to the partSelect element
- Input: `physicalPortID` of type ***String*** - Handle to a physicalPort element

F.7.69.6 getPhysicalPortSubPortIDs

Description: Returns the handles to all the subPort defined on the given physicalPort element.

- Returns: `subPortIDs` of type ***String*** - List of handles to the subPort elements
- Input: `physicalPortID` of type ***String*** - Handle to a physicalPort element

F.7.69.7 getPortMapIsInformative

Description: Returns the `isInformative` defined on the given portMap element.

- Returns: `value` of type ***Boolean*** - True if portmap should be nestlisted, false otherwise
- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.69.8 getPortMapLogicalPortID

Description: Returns the handle to the logicalPort defined on the given portMap element.

- Returns: `logicalPortID` of type ***String*** - Handle to the logicalPort
- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.69.9 getPortMapLogicalTieOff

Description: Returns the logicalTieOff defined on the given portMap element.

- Returns: `value` of type ***Long*** - The logical port tieOff value
- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.69.10 getPortMapLogicalTieOffExpression

Description: Returns the logicalTieOff expression defined on the given portMap element.

- Returns: `expression` of type ***String*** - The logical port tieOff expression
- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.69.11 getPortMapLogicalTieOffID

Description: Returns the handle to the LogicalTieOff defined on the given portMap element.

- Returns: `logicalTieOffID` of type ***String*** - Handle to the LogicalTieOff element
- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.69.12 getPortMapPhysicalPortID

Description: Returns the handle to the physicalPort defined on the given portMap element.

- Returns: `physicalPortID` of type ***String*** - Handle to the physicalPort
- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.70 Port map (EXTENDED)

F.7.70.1 addPhysicalPortSubPort

Description: Adds a subPort to the given physicalPort element

- Returns: subPortID of type **String** - Handle to a subPort element
- Input: physicalPortID of type **String** - Handle to a physicalPort element
- Input: name of type **String** - The subPort name

F.7.70.2 addPortMapPhysicalPortSubPort

Description: Adds a subPort to the containing element.

- Returns: subPortID of type **String** - Handle to the added subPort
- Input: portMapID of type **String** - Handle to a portMap element
- Input: name of type **String** - Name of the subPort

F.7.70.3 removePhysicalPortPartSelect

Description: Removes a partSelect on the given portMap physicalPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: physicalPortID of type **String** - Handle to a portMap physicalPort element

F.7.70.4 removePhysicalPortSubPort

Description: Removes the given subPort element from a physicalPort

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subPortID of type **String** - Handle to a subPort element

F.7.70.5 removePortMapIsInformative

Description: Removes the isInformative from the given portMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portMapID of type **String** - Handle to a portMap element

F.7.70.6 setAbstractionTypeAbstractionRef

Description: Sets the abstraction reference on the given abstractionType element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: abstractionTypeID of type **String** - Handle to an abstractionType element
- Input: vlnv of type **String[]** - The abstractionType reference

F.7.70.7 setLogicalPortRange

Description: Sets the range for the given logicalPort element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: logicalPortID of type **String** - Handle to a logicalPort element
- Input: range of type **String[]** - leftExpression at range index 0 & rightExpression at range 1

F.7.70.8 setPhysicalPortPartSelect

Description: Sets a partSelect on the given portMap physicalPort element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: physicalPortID of type **String** - Handle to a portMap physicalPort element
- Input: range of type **String[]** - Handle to of type String set the minimum (range[0]) and the maximum (range[1]), on the partSelect
- Input: indices of type **String[]** - Handle to values of type String. Set all the index on the partSelect

F.7.70.9 setPortMapIsInformative

Description: Sets the isInformative for the given portMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portMapID of type **String** - Handle to a portMap element
- Input: value of type **Boolean** - IsInformative value

F.7.70.10 setPortMapLogicalPort

Description: Sets the logicalPort on the given portMap element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portMapID of type **String** - Handle to a portMap element
- Input: name of type **String** - The logicalPort name

F.7.70.11 setPortMapLogicalTieOff

Description: Sets the logicalTieOff with the given value for the given portMap element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portMapID of type **String** - Handle to an portMap element
- Input: expression of type **String** - logicalTieOff value

F.7.70.12 setPortMapPhysicalPort

Description: Sets the physicalPort on the given portMap element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portMapID of type **String** - Handle to a portMap element
- Input: name of type **String** - The physicalPort name

F.7.70.13 setSubPortMapPartSelect

Description:

- Returns: partSelectID of type **String** - Handle to a partSelect element
- Input: subPortID of type **String** - Handle to a subPort element
- Input: range of type **String[]** - Create the range on the partSelect with “left” for range[0] and “right” for range[1]. Set to null if you only want indices.
- Input: indices of type **String[]** - Handle to values of type String. Set all the index on the partSelect

F.7.71 Power (BASE)

F.7.71.1 getPortTransactionalPowerConstraintIDs

Description: Returns the handle to the powerConstraint defined on the given port transactional element.

- Returns: powerConstraintIDs of type **String** - List of handles to the powerConstraint elements
- Input: transactionalID of type **String** - Handle to a port transactional element

F.7.71.2 getPortWirePowerConstraintIDs

Description: Returns the handle to the powerConstraint defined on the given port wire element.

- Returns: powerConstraintIDs of type **String** - List of handles to the powerConstraint elements
- Input: wireID of type **String** - Handle to a port wire element

F.7.71.3 getPowerConstraintPowerDomainRefByID

Description: Returns the handle to the powerDomain referenced from the given powerConstraint element.

- Returns: powerDomainID of type **String** - Handle of the referenced power domain
- Input: powerConstraintID of type **String** - Handle to a powerConstraint element

F.7.71.4 getPowerConstraintPowerDomainRefByName

Description: Returns the powerDomain referenced from the given powerConstraint element.

- Returns: powerDomainRef of type **String** - The referenced powerDomain
- Input: powerConstraintID of type **String** - Handle to a powerConstraint element

F.7.71.5 getPowerConstraintRange

Description: Returns the range of the given powerConstraint.

- Returns: values of type **String** - Array of two range values: left and right
- Input: powerConstraintID of type **String** - Handle to a powerConstraint element

F.7.71.6 getPowerConstraintRangeLeftID

Description: Returns the handle to the left range of the given powerConstraint.

- Returns: leftID of type **String** - Handle to the left range
- Input: powerConstraintID of type **String** - Handle to a powerConstraint element

F.7.71.7 getPowerConstraintRangeRightID

Description: Returns the handle to the right range of the given powerConstraint.

- Returns: rightID of type **String** - Handle to the right range
- Input: powerConstraintID of type **String** - Handle to a powerConstraint element

F.7.71.8 getPowerDomainAlwaysOn

Description: Returns the alwaysOn resolved value defined on the given powerDomain element.

- Returns: value of type **Boolean** - True is the powerDomain is always on
- Input: powerDomainID of type **String** - Handle to a powerDomain element

F.7.71.9 getPowerDomainAlwaysOnExpression

Description: Returns the alwaysOn expression defined on the given powerDomain element.

- Returns: expression of type **String** - The expression on the alwaysON element
- Input: powerDomainID of type **String** - Handle to a powerDomain element

F.7.71.10 getPowerDomainAlwaysOnID

Description: Returns the alwaysOn identifier defined on the given powerDomain element.

- Returns: alwaysOnID of type **String** - Handle to an alwaysOn element
- Input: powerDomainID of type **String** - Handle to a powerDomain element

F.7.71.11 getPowerDomainName

Description: Returns the name of the given PowerDomain element.

- Returns: name of type **String** - The powerDomain name
- Input: powerDomainID of type **String** - Handle to a powerDomain element

F.7.71.12 getPowerDomainSubDomainOf

Description: Returns the parent power domain defined on the given powerDomain element.

- Returns: subDomainOf of type **String** - The referenced parent powerDomain
- Input: powerDomainID of type **String** - Handle to a powerDomain element

F.7.71.13 getPowerDomainSubDomainOfRefByID

Description: Returns the handle to the parent powerDomain referenced from the given powerDomain element.

- Returns: powerDomainID of type **String** - Handle to the referenced powerDomain element
- Input: powerDomainID of type **String** - Handle to a powerDomain element

F.7.72 Power (EXTENDED)

F.7.72.1 addComponentInstancePowerDomainLink

Description: Adds a Power Domain Link to the containing element.

- Returns: powerDomainLinkID of type **String** - the powerDomainLink identifier
- Input: componentInstanceID of type **String** - Handle to a componentInstance
- Input: externalPowerDomainRef of type **String** - Handle to the reference of a powerDomain on the top Component
- Input: internalPowerDomainRef of type **String** - Handle to the reference of a powerDomain defined on an instance

F.7.72.2 addComponentPowerDomain

Description: Adds a powerDomain to the given component.

- Returns: powerDomainID of type **String** - the new ID of the created powerDomain
- Input: componentID of type **String** - Handle to an component
- Input: name of type **String** - Handle to the name of a component

F.7.72.3 addPortTransactionalPowerConstraint

Description: Adds a powerConstraint to a transactional Port.

- Returns: powerConstraintID of type **String** - Handle to the added powerConstraint
- Input: transactionalID of type **String** - Handle to transactional element
- Input: powerDomainRef of type **String** - Handle to the reference of a powerDomain on the powerConstraint

F.7.72.4 addPortWirePowerConstraint

Description: Adds a powerConstraint to a wire Port.

- Returns: powerConstraintID of type **String** - Handle to the added powerConstraint
- Input: wireID of type **String** - Handle to wire element
- Input: powerDomainRef of type **String** - Handle to the reference of a powerDomain on the powerConstraint

F.7.72.5 addPowerDomainLinkInternalPowerDomainReference

Description: Adds an internalPowerDomainReference on a powerDomainLink element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element
- Input: internalPowerDomainReference of type **String** - the internal power reference to be added

F.7.72.6 removeComponentInstancePowerDomainLink

Description: Removes the given PowerDomainRef from its containing componentInstance element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerDomainRefID of type **String** - Handle to a powerDomainRef

F.7.72.7 removeComponentPowerDomain

Description: Removes the given powerDomain from its containing component object.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerDomainID of type **String** - Handle to a powerDomain

F.7.72.8 removePowerConstraintRange

Description: Removes the range for the given PowerConstraint.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerConstraintID of type **String** - Handle to a powerConstraint

F.7.72.9 removePowerDomainLinkInternalPowerDomainRef

Description: Removes an internalPowerReference on a powerDomainLink.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerDomainLinkID of type **String** - Handle to a powerDomainLink element
- Input: internalPowerDomainRef of type **String** - The internalPowerDomainRef to remove

F.7.72.10 removePowerDomainSubDomainOf

Description: Removes the alwaysOn boolean on a powerDomain element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerDomainID of type **String** - Handle to a powerDomain element

F.7.72.11 setPowerConstraintPowerDomainRef

Description: Sets the powerDomainRef of the given powerConstraint.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerConstraintID of type **String** - Handle to a powerConstraint element
- Input: powerDomainRef of type **String** - Name of the referenced powerDomain

F.7.72.12 setPowerConstraintRange

Description: Sets the range for the given powerConstraint.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerConstraintID of type **String** - Handle to a powerConstraint
- Input: range of type **String[]** - Range, defined as a pair of left and right value expressions

F.7.72.13 setPowerDomainAlwaysOn

Description: Sets the alwaysOn boolean on a powerDomain element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerDomainID of type **String** - Handle to a powerDomain element
- Input: alwaysOn of type **String** - the always on value to set

F.7.72.14 setPowerDomainSubDomainOf

Description: Sets the alwaysOn boolean on a powerDomain element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: powerDomainID of type **String** - Handle to a powerDomain element
- Input: subDomainOf of type **String** - the subDomainOf on value to set

F.7.73 Register (BASE)

F.7.73.1 getAliasOfAlternateRegisterRefByName

Description: Returns the alternateRef defined on the given aliasOf element.

- Returns: alternateRegisterRef of type **String** - The referenced alternateRegister
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.73.2 getAliasOfAlternateRegisterRefID

Description: Returns the handle to the alternateRegisterRef defined on the given aliasOf element.

- Returns: alternateRegisterRefID of type **String** - Handle to the alternateRegisterRef element
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.73.3 getAliasOfFieldRefByName

Description: Returns the fieldRef defined on the given aliasOf element.

- Returns: `fieldRef` of type ***String*** - The referenced register field
- Input: `aliasOfID` of type ***String*** - Handle to an aliasOf element

F.7.73.4 getAliasOfFieldRefID

Description: Returns the handle to the fieldRef defined on the given aliasOf element.

- Returns: `fieldRefID` of type ***String*** - Handle to the fieldRef element
- Input: `aliasOfID` of type ***String*** - Handle to an aliasOf element

F.7.73.5 getAliasOfRegisterRefByName

Description: Returns the registerRef defined on the given aliasOf element.

- Returns: `registerRef` of type ***String*** - The referenced register
- Input: `aliasOfID` of type ***String*** - Handle to an aliasOf element

F.7.73.6 getAliasOfRegisterRefID

Description: Returns the handle to the registerRef defined on the given aliasOf element.

- Returns: `registerRefID` of type ***String*** - Handle to the registerRef element
- Input: `aliasOfID` of type ***String*** - Handle to an aliasOf element

F.7.73.7 getAlternateRegisterFieldIDs

Description: Returns the handles to all the fields defined on the given alternateRegister element.

- Returns: `regFieldIDs` of type ***String*** - List of handles to the register field elements
- Input: `alternateRegisterID` of type ***String*** - Handle to an alternateRegister element

F.7.73.8 getAlternateRegisterModeRefIDs

Description: Returns the handles to all the modeRefs defined on the given alternateRegister.

- Returns: `modeRefIDs` of type ***String*** - List of handles to the modeRef elements
- Input: `alternateRegisterID` of type ***String*** - Handle to an alternateRegister element

F.7.73.9 getAlternateRegisterRefAlternateRegisterRefByName

Description: Returns the alternateRegisterRef defined on the given alternateRegisterRef element.

- Returns: `alternateRegisterRef` of type ***String*** - The referenced alternateRegister
- Input: `alternateRegisterRefID` of type ***String*** - Handle to an alternateRegisterRef element

F.7.73.10 getAlternateRegisterTypeIdentifier

Description: Returns the typeIdentifier defined on the given alternateRegister element.

- Returns: `typeIdentifier` of type ***String*** - The typeIdentifier value
- Input: `alternateRegisterID` of type ***String*** - Handle to an alternateRegister element

F.7.73.11 getAlternateRegisterVolatility

Description: Returns the volatile value defined on the given alternateRegister element.

- Returns: `volatile` of type ***Boolean*** - The volatile value
- Input: `alternateRegisterID` of type ***String*** - Handle to an alternateRegister element

F.7.73.12 getBroadcastToAddressBlockRefByName

Description: Returns the addressBlockRef defined on the given broadcastTo element.

- Returns: `addressBlockRef` of type ***String*** - The referenced addressBlock
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.13 getBroadcastToAddressBlockRefID

Description: Returns the handle to the addressBlock defined on the given broadcastTo element.

- Returns: `addressBlockID` of type ***String*** - Handle to the addressBlock element
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.14 getBroadcastToAddressSpaceRefByName

Description: Returns the addressSpaceRef defined on the given broadcastTo element.

- Returns: `addressSpaceRef` of type ***String*** - The referenced addressSpace
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.15 getBroadcastToAddressSpaceRefID

Description: Returns the handle to the addressSpaceRef defined on the given broadcastTo element.

- Returns: `addressSpaceRefID` of type ***String*** - Handle to the addressSpaceRef element
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.16 getBroadcastToAlternateRegisterRefByName

Description: Returns the alternateRegisterRef defined on the given broadcastTo element.

- Returns: `alternateRegisterRef` of type ***String*** - The referenced alternateRegister
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.17 getBroadcastToAlternateRegisterRefID

Description: Returns the handle to the alternateRegisterRef defined on the given fieldSlice element.

- Returns: `alternateRegisterRefID` of type ***String*** - Handle to the alternateRegisterRef element
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.18 getBroadcastToBankRefByNames

Description: Returns all the bankRefs defined on the given broadcastTo element.

- Returns: `bankRef` of type ***String*** - The list of all bankRef values
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.19 getBroadcastToBankRefIDs

Description: Returns the handles to all the bankRefs defined on the given broadcastTo element.

- Returns: `bankRefIDs` of type ***String*** - List of handles to the bankRef elements
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.20 getBroadcastToFieldRefByName

Description: Returns the fieldRef defined on the given broadcastTo element.

- Returns: `fieldRef` of type ***String*** - The referenced field
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.21 getBroadcastToFieldRefID

Description: Returns the fieldRef defined on the given broadcastTo element.

- Returns: `fieldRefID` of type ***String*** - Handle to the fieldRef element
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.22 getBroadcastToMemoryMapRefByName

Description: Returns the memoryMapRef defined on the given broadcastTo element.

- Returns: `memoryMapRef` of type ***String*** - The referenced memoryMap
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.23 getBroadcastToMemoryMapRefID

Description: Returns the handle to the memoryMapRef defined on the given broadcastTo element.

- Returns: `memoryMapRefID` of type ***String*** - Handle to the memoryMapRef element
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.24 getBroadcastToRegisterFileRefByNames

Description: Returns all the registerFileRefs defined on the given broadcastTo element.

- Returns: `registerFileRefs` of type ***String*** - List of the referenced registerFiles
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.25 getBroadcastToRegisterFileRefIDs

Description: Returns all the registerFileRefs defined on the given broadcastTo element.

- Returns: `registerFileRefIDs` of type ***String*** - The list of all registerFileRef values
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.26 getBroadcastToRegisterRefByName

Description: Returns the registerRef defined on the given broadcastTo element.

- Returns: `registerRef` of type ***String*** - The referenced register
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.27 getBroadcastToRegisterRefID

Description: Returns the handle to the registerRef defined on the given broadcastTo element.

- Returns: `registerRefID` of type ***String*** - Handle to the RegisterRef element
- Input: `broadcastToID` of type ***String*** - Handle to a broadcastTo element

F.7.73.28 getEnumeratedValueExpression

Description: Returns the expression defined on the given enumerationValue element.

- Returns: `expression` of type ***String*** - The enumerationValue expression
- Input: `enumeratedValueID` of type ***String*** - Handle to an enumerationValue element

F.7.73.29 getEnumeratedValueUsage

Description: Returns the usage defined on the given enumeratedValue element.

- Returns: `usage` of type ***String*** - The usage
- Input: `enumeratedValueID` of type ***String*** - Handle to an enumeratedValue element

F.7.73.30 getEnumeratedValueValue

Description: Returns the value defined on the given enumeratedValue element.

- Returns: `value` of type ***Long*** - The enumeratedValue value
- Input: `enumeratedValueID` of type ***String*** - Handle to an enumeratedValue element

F.7.73.31 getEnumeratedValueValueExpression

Description: Returns the valueExpression defined on the given enumeratedValue element.

- Returns: `valueExpression` of type ***String*** - The enumerated value expression
- Input: `enumeratedValueID` of type ***String*** - Handle to an enumeratedValue element

F.7.73.32 getEnumeratedValueValueID

Description: Returns the handle to the value defined on the given enumeratedValue element.

- Returns: `valueID` of type ***String*** - Handle to the value element
- Input: `enumeratedValueID` of type ***String*** - Handle to an enumeratedValue element

F.7.73.33 getEnumeratedValuesEnumeratedValueIDs

Description: Returns the handles to all the enumeratedValues defined on the given enumeratedValues element

- Returns: `enumeratedValueIDs` of type ***String*** - List of handles to the enumeratedValue elements
- Input: `enumeratedValuesID` of type ***String*** - Handle to an enumeratedValues element

F.7.73.34 getEnumeratedValuesEnumerationDefinitionRefByExternalTypeDefID

Description: Returns the handle to the externalTypeDefinitions referenced by the typeDefinitions attribute of the enumerationDefinitionRef defined on the given enumeratedValues element.

- Returns: `externalTypeDefID` of type ***String*** - Handle to the externalTypeDefinitions element referenced by the typeDefinitions attribute

- Input: enumeratedValuesID of type **String** - Handle to an enumeratedValues element

F.7.73.35 getEnumeratedValuesEnumerationDefinitionRefByID

Description: Returns the handle to the enumerationDefinition referenced from the given enumeratedValues element

- Returns: enumerationDefinitionID of type **String** - Handle to the referenced enumerationDefinition element
- Input: enumeratedValuesID of type **String** - Handle to an enumeratedValues element

F.7.73.36 getEnumeratedValuesEnumerationDefinitionRefByName

Description: Returns the enumerationDefinitionRefs defined on the given enumeratedValues element.

- Returns: enumerationDefinitionRef of type **String** - The referenced enumerationDefinition
- Input: enumeratedValuesID of type **String** - Handle to an enumeratedValues element

F.7.73.37 getEnumeratedValuesEnumerationDefinitionRefID

Description: Returns the handle to the enumerationDefinitionRef defined on the given enumeratedValues element.

- Returns: enumerationDefinitionRefID of type **String** - Handle to the enumerationDefinitionRef element
- Input: enumeratedValuesID of type **String** - Handle to an enumeratedValues element

F.7.73.38 getFieldBitOffsetID

Description: Returns the handle to the bitOffset defined on the given field element.

- Returns: bitOffsetID of type **String** - Handle to the bitOffset element
- Input: fieldID of type **String** - Handle to a field element

F.7.73.39 getFieldDefinitionBitWidthID

Description: Returns the handle to the bitWidth defined on the given fieldDefinition element.

- Returns: bitWidthID of type **String** - Handle to the bitWidth element
- Input: fieldDefinitionID of type **String** - Handle to a fieldDefinition element

F.7.73.40 getFieldDefinitionEnumeratedValueIDs

Description: Returns the handles to all the enumeratedValues defined on the given fieldDefinition element.

- Returns: enumeratedValueIDs of type **String** - List of handles to the enumeratedValue elements
- Input: fieldDefinitionID of type **String** - Handle to a fieldDefinition element

F.7.73.41 getFieldDefinitionTypeIdentifier

Description: Returns the typeIdentifier value of the given fieldDefinition element.

- Returns: typeIdentifier of type **String** - The value of the typeIdentifier defined on the fieldDefinition
- Input: fieldDefinitionID of type **String** - Handle to a fieldDefinition element

F.7.73.42 getFieldDefinitionVolatile

Description: Returns the volatile Boolean of the given fieldDefinition.

- Returns: `volatile` of type ***Boolean*** - True if the field is volatile
- Input: `fieldDefinitionID` of type ***String*** - Handle to a fieldDefinition element

F.7.73.43 getFieldRefFieldRefByName

Description: Returns the fieldRef defined on the given fieldRef element.

- Returns: `fieldRef` of type ***String*** - The referenced field
- Input: `fieldRefID` of type ***String*** - Handle to a fieldRef element

F.7.73.44 getFieldRefIndexIDs

Description: Returns the handles to all the index elements defined on the given fieldRef element.

- Returns: `indexIDs` of type ***String*** - List of handles to the index elements
- Input: `fieldRefID` of type ***String*** - Handle to a fieldRef element

F.7.73.45 getRegisterAddressOffset

Description: Returns the addressOffset value defined on the given register element.

- Returns: `addressOffset` of type ***Long*** - The address offset
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.46 getRegisterAddressOffsetExpression

Description: Returns the addressOffset expression defined on the given register element.

- Returns: `addressOffsetExpression` of type ***String*** - The addressOffset expression
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.47 getRegisterAddressOffsetID

Description: Returns the handle to the AddressOffset defined on the given register element.

- Returns: `addressOffsetID` of type ***String*** - Handle to the addressOffset element
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.48 getRegisterAlternateRegisterIDs

Description: Returns the handles to all the alternativeRegisters defined on the given register element.

- Returns: `alternateRegisterIDs` of type ***String*** - List of handles to the alternateRegister elements
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.49 getRegisterFieldAliasOfID

Description: Returns the handle to the aliasOf defined on the given register field element.

- Returns: `aliasOfID` of type ***String*** - Handle to the aliasOf element
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.73.50 getRegisterFieldBitOffset

Description: Returns the bitOffset in the given register field element.

- Returns: `offset` of type **Long** - Field bit offset
- Input: `registerFieldID` of type **String** - Handle to a register field element

F.7.73.51 getRegisterFieldBitOffsetExpression

Description: Returns the bitOffset expression in the given register field element.

- Returns: `offsetExpression` of type **String** - The bitOffset expression
- Input: `registerFieldID` of type **String** - Handle to a register field element

F.7.73.52 getRegisterFieldBitOffsetID

Description: Returns the handle to the bitOffset defined on the given register field element.

- Returns: `bitOffsetID` of type **String** - Handle to the bitOffset element
- Input: `registerFieldID` of type **String** - Handle to a register field element

F.7.73.53 getRegisterFieldBitWidth

Description: Returns the bitWidth element defined on the given register field element.

- Returns: `width` of type **Long** - The field width in bits
- Input: `registerFieldID` of type **String** - Handle to a register field element

F.7.73.54 getRegisterFieldBitWidthExpression

Description: Returns the bitWidth defined on the given register field element.

- Returns: `width` of type **String** - The bitWidth expression
- Input: `registerFieldID` of type **String** - Handle to a register field element

F.7.73.55 getRegisterFieldBitWidthID

Description: Returns the handle to the width defined on the given register field element.

- Returns: `widthID` of type **String** - Handle to the width element
- Input: `registerFieldID` of type **String** - Handle to a register field element

F.7.73.56 getRegisterFieldEnumeratedValuesID

Description: Returns the handle to the enumeratedValues defined on the given register field element.

- Returns: `enumeratedValuesID` of type **String** - Handle to the enumeratedValues element
- Input: `registerFieldID` of type **String** - Handle to a register field

F.7.73.57 getRegisterFieldFieldDefinitionRefByExternalTypeDefID

Description: Returns the handle to the externalTypeDefinitions referenced by the typeDefinitions attribute of the fieldDefinitionRef defined on the given register field element.

- Returns: `fieldDefinitionID` of type **String** - Handle to the externalTypeDefinitions element referenced by the typeDefinitions attribute
- Input: `registerFieldID` of type **String** - Handle to a register field element

F.7.73.58 getRegisterFieldFieldDefinitionRefByID

Description: Returns the handle to the fieldDefinition (defined in the typeDefinitions root object) referenced from the given register field element.

- Returns: `fieldDefinitionID` of type ***String*** - Handle to the referenced fieldDefinition
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.73.59 getRegisterFieldFieldDefinitionRefByName

Description: Returns the fieldDefinitionRef defined on the given register field element.

- Returns: `fieldDefinitionRef` of type ***String*** - The referenced fieldDefinition
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.73.60 getRegisterFieldFieldDefinitionRefID

Description: Returns the handle to the fieldDefinitionRef element defined on the given register field element.

- Returns: `fieldDefinitionRefID` of type ***String*** - Handle to the fieldDefinitionRef element
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.73.61 getRegisterFieldIDs

Description: Returns the handles to all the fields defined on the given register element.

- Returns: `regFieldIDs` of type ***String*** - List of handles to the register field elements
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.62 getRegisterFieldResetIDs

Description: Returns the handles to all the reset elements defined on the given register field element.

- Returns: `resetIDs` of type ***String*** - List of handles to the reset elements
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.73.63 getRegisterFieldTypeIdentifier

Description: Returns the typeIdentifier element defined on the given register field element.

- Returns: `typeIdentifier` of type ***String*** - The type identifier
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.73.64 getRegisterFieldVolatility

Description: Returns the volatile value defined on the given register field element.

- Returns: `volatile` of type ***Boolean*** - The volatile value
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.73.65 getRegisterFileAddressOffsetID

Description: Returns the handle to the addressOffset defined on the given registerFile element.

- Returns: `addressOffsetID` of type ***String*** - Handle to the addressOffset element
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.73.66 getRegisterRefAttributeByID

Description: Returns the handle to register referenced from the given registerRef element.

- Returns: `registerID` of type ***String*** - Handle to the referenced register element
- Input: `registerRefID` of type ***String*** - Handle to the registerRef element

F.7.73.67 getRegisterRefIndexIDs

Description: Returns the handles to all the index elements defined on the given registerRef element.

- Returns: `indexIDs` of type ***String*** - List of handles to the index elements
- Input: `registerRefID` of type ***String*** - Handle to a registerRef element

F.7.73.68 getRegisterRefRegisterRefByName

Description: Returns the registerRef defined on the given registerRef element.

- Returns: `registerRef` of type ***String*** - The referenced register
- Input: `registerRefID` of type ***String*** - Handle to a registerRef element

F.7.73.69 getRegisterRegisterDefinitionRefByExternalTypeDefID

Description: Returns the handle to the externalTypeDefinitions referenced by the typeDefinitions attribute of the registerDefinitionRef defined on the given register element.

- Returns: `externalTypeDefinitionsID` of type ***String*** - Handle to the externalTypeDefinitions element referenced by the typeDefinitions attribute
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.70 getRegisterRegisterDefinitionRefByID

Description: Returns the handle to the registerDefinition (defined in the typeDefinitions root object) referenced from the given register element.

- Returns: `registerDefinitionID` of type ***String*** - Handle to the referenced registerDefinition
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.71 getRegisterRegisterDefinitionRefByName

Description: Returns the registerDefinitionRef defined on the given register element.

- Returns: `registerDefinitionRef` of type ***String*** - The referenced registerDefinition
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.72 getRegisterRegisterDefinitionRefID

Description: Returns the handle to the registerDefinitionRef element defined on the register element.

- Returns: `registerDefinitionRefID` of type ***String*** - Handle to the registerDefinitionRef element
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.73 getRegisterSize

Description: Returns the size value defined on the given register element.

- Returns: `size` of type ***Long*** - The size value

- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.74 getRegisterSizeExpression

Description: Returns the size expression defined on the given register element.

- Returns: `size` of type ***String*** - The register size expression
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.75 getRegisterSizeID

Description: Returns the handle to the size defined on the given register element.

- Returns: `sizeID` of type ***String*** - Handle to the size element
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.76 getRegisterTypeIdentifier

Description: Returns the typeIdentifier defined on the given register element.

- Returns: `typeIdentifier` of type ***String*** - The typeIdentifier value
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.77 getRegisterVolatility

Description: Returns the volatile value defined on the given register element.

- Returns: `volatile` of type ***Boolean*** - The volatile value
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.78 getResetMask

Description: Returns the mask (resolved) value defined on the given reset element.

- Returns: `mask` of type ***Long*** - The mask value
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.73.79 getResetMaskExpression

Description: Returns the mask expression defined on the given reset element.

- Returns: `expression` of type ***String*** - The mask expression
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.73.80 getResetMaskID

Description: Returns the handle to the mask defined on the given reset element.

- Returns: `maskID` of type ***String*** - Handle to the mask element
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.73.81 getResetValue

Description: Returns the (resolved) value defined on the given reset element.

- Returns: `value` of type ***Long*** - The reset value
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.73.82 getResetValueExpression

Description: Returns the value expression defined on the given reset element.

- Returns: expression of type **String** - The value expression
- Input: resetID of type **String** - Handle to a reset element

F.7.73.83 getResetValueID

Description: Returns the handle to the value defined on the given reset element.

- Returns: valueID of type **String** - Handle to the reset value
- Input: resetID of type **String** - Handle to a reset element

F.7.74 Register (EXTENDED)

F.7.74.1 addAlternateRegisterField

Description: Adds regField with the given name, offset, and width to the given alternateRegister element.

- Returns: regFieldID of type **String** - Handle to a new regField element
- Input: alternateRegisterID of type **String** - Handle to an alternateRegister element
- Input: name of type **String** - RegField name
- Input: offset of type **String** - RegField offset
- Input: width of type **String** - RegField width

F.7.74.2 addAlternaterRegisterModeRef

Description: Adds a modeRef to the given alternateRegister.

- Returns: modeRefID of type **String** - the modeRef identifier
- Input: alternateRegisterID of type **String** - Handle to an alternateRegister element
- Input: modeRef of type **String** - Name of the referenced mode
- Input: priority of type **Long** - The priority of the added mode reference

F.7.74.3 addBroadcastToBankRef

Description: Adds a bankRef on a broadcastTo element.

- Returns: bankRefID of type **String** - Handle to a bankRef identifier
- Input: broadcastToID of type **String** - Handle to an broadcast element
- Input: bankRef of type **String** - value of the bankRef to add

F.7.74.4 addBroadcastToRegisterFileRef

Description: Adds a registerRef on a broadcastTo element.

- Returns: registerFileRefID of type **String** - Handle to the added registerFileRef
- Input: broadcastToID of type **String** - Handle to a broadcastTo element
- Input: registerFileRef of type **String** - Name of the referenced registerFile

F.7.74.5 addEnumeratedValuesFieldEnumeratedValue

Description: Adds an enumeratedValue on the given enumeratedValues element

- Returns: enumeratedValueID of type **String** - Handle to a new enumeratedValue element
- Input: enumeratedValuesID of type **String** - Handle to an enumeratedValues element
- Input: name of type **String** - EnumeratedValue name
- Input: value of type **String** - EnumeratedValue value

F.7.74.6 addFieldRefIndex

Description: Adds an index to an fieldRef element.

- Returns: indexID of type **String** - Handle to the added index
- Input: fieldRefID of type **String** - Handle to a fieldRef element
- Input: value of type **String** - Index value

F.7.74.7 addRegisterAlternateRegister

Description: Adds an alternateRegister to the given register.

- Returns: alternateRegisterID of type **String** - Handle to the added alternateRegister
- Input: registerID of type **String** - Handle to a register element
- Input: name of type **String** - the name of the alternateRegister
- Input: modeRef of type **String** - the modeRef to set on the alternate register
- Input: priority of type **Long** - the priority of the modeRef
- Input: fieldName of type **String** - the field name on the alternateRegister
- Input: fieldOffset of type **String** - the Offset of the field on the alternateRegister
- Input: fieldWidth of type **String** - the width of the field on the alternateRegister

F.7.74.8 addRegisterField

Description: Adds regField with the given name, offset, and width to the given register element.

- Returns: regFieldID of type **String** - Handle to a new regField element
- Input: registerID of type **String** - Handle to a register element
- Input: name of type **String** - RegField name
- Input: offset of type **String** - RegField offset
- Input: width of type **String** - RegField width

F.7.74.9 addRegisterFieldReset

Description: Adds a reset with the given type and value to the given regField element.

- Returns: resetID of type **String** - Handle to the added reset
- Input: registerFieldID of type **String** - Handle to a regField element
- Input: value of type **String** - Reset value expression

F.7.74.10 addRegisterRefIndex

Description: Adds an index to a registerRef element.

- Returns: indexID of type **String** - Handle to the added index
- Input: registerRefID of type **String** - Handle to a registerRef element
- Input: value of type **String** - Index value

F.7.74.11 removeAliasOfAlternateRegisterRef

Description: Removes an alternateRegisterRef on an aliasOf element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle of an aliasOf element

F.7.74.12 removeAliasOfRegisterRef

Description: Removes the registerRef on an aliasOf of a field element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.74.13 removeAlternateRegisterField

Description: Removes the given regField element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: regFieldID of type **String** - Handle to a regField element

F.7.74.14 removeAlternateRegisterTypeIdentifier

Description: Removes typeIdentifier from the given alternateRegister element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: alternateRegisterID of type **String** - Handle to an alternateRegister element

F.7.74.15 removeAlternateRegisterVolatility

Description: Removes volatility from the given alternateRegister element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: alternateRegisterID of type **String** - Handle to an alternateRegister element

F.7.74.16 removeAlternateRegisterModeRef

Description: Removes the given modeRef from its containing element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeRefID of type **String** - Handle to a modeRef element

F.7.74.17 removeBroadcastToAddressBlockRef

Description: Removes the AddressBlock for a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to a broadcastTo element

F.7.74.18 removeBroadcastToAlternateRegisterRef

Description: Removes an alternateRegisterRef on a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle of a broadcastTo element

F.7.74.19 removeBroadcastToBankRef

Description: Removes the given bankRef from its containing broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankRefID of type **String** - Handle a bankRef element

F.7.74.20 removeBroadcastToMemoryMapRef

Description: Removes the memoryMapRef for a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to a broadcastTo element

F.7.74.21 removeBroadcastToRegisterFileRef

Description: Removes the given registerRef from its containing broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerRefID of type **String** - Handle to a registerRef element

F.7.74.22 removeBroadcastToRegisterRef

Description: Removes the RegisterRef for a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to a broadcastTo element

F.7.74.23 removeEnumeratedValuesEnumValue

Description: Removes the given enumValue element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: enumValueID of type **String** - Handle to an enumValue element

F.7.74.24 removeFieldRefIndex

Description: Removes the given index from its containing fieldRef element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: indexID of type **String** - Handle to an index element

F.7.74.25 removeRegisterAlternateRegister

Description: Removes the given alternateRegister element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: alternateRegisterID of type **String** - Handle to an alternateRegister element

F.7.74.26 removeRegisterField

Description: Removes the given register field element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFieldID of type **String** - Handle to a field of a register element

F.7.74.27 removeRegisterFieldArray

Description: Removes the array element from the given register field.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFieldID` of type ***String*** - Handle to a registerField element

F.7.74.28 removeRegisterFieldBitWidth

Description: Removes the bitWidth element from the given register field.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFieldID` of type ***String*** - Handle to a registerField element

F.7.74.29 removeRegisterFieldEnumeratedValues

Description: Removes the enumeratedValues element from a field of a register

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFieldID` of type ***String*** - Handle to a register Field element

F.7.74.30 removeRegisterFieldFieldAccessPolicies

Description: Removes the fieldAccessPolicies on the given field on a register

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFieldID` of type ***String*** - Handle to an field on a register element

F.7.74.31 removeRegisterFieldReset

Description: Removes the given reset element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.74.32 removeRegisterFieldTypeIdentifier

Description: Removes the typeIdentifier from the given register field element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.74.33 removeRegisterFieldVolatility

Description: Removes the volatility from the given register field element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFieldID` of type ***String*** - Handle to a register field element

F.7.74.34 removeRegisterRefIndex

Description: Removes the given index from its containing registerRef element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indexID` of type ***String*** - Handle to an index element

F.7.74.35 removeRegisterTypeIdentifier

Description: Removes typeIdentifier from the given register element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerID of type **String** - Handle to a register element

F.7.74.36 removeRegisterVolatility

Description: Removes volatility from the given register element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerID of type **String** - Handle to a register element

F.7.74.37 removeResetMask

Description: Removes the mask from the given reset element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: resetID of type **String** - Handle to a reset element

F.7.74.38 removeUsageAttribute

Description: Removes the value of the usage field of an EnumeratedValue

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: enumeratedValueID of type **String** - Handle to an enumeratedValue element

F.7.74.39 setAliasOfAlternateRegisterRef

Description: Sets an alternateRegisterRef on an aliasOf element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle of an aliasOf element
- Input: alternateRegisterRef of type **String** - the alternateRegisterRef value to set

F.7.74.40 setAliasOfFieldRef

Description: Sets the fieldRef on the aliasOf of a field element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle to an aliasOf element
- Input: fieldRef of type **String** - Name of the referenced field

F.7.74.41 setAliasOfRegisterRef

Description: Sets the registerRef on an aliasOf of a field element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: aliasOfID of type **String** - Handle to an aliasOf element
- Input: registerRef of type **String** - Name of the referenced register

F.7.74.42 setAlternateRegisterTypeIdentifier

Description: Sets typeIdentifier with the given value for the given alternateRegister element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: alternateRegisterID of type **String** - Handle to an alternateRegister element
- Input: typeIdentifier of type **String** - The alternateRegister typeIdentifier

F.7.74.43 setAlternateRegisterVolatility

Description: Sets volatility with the given value for the given alternateRegister element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: alternateRegisterID of type **String** - Handle to an alternateRegister element
- Input: _volatile of type **Boolean** - The alternateRegister volatility

F.7.74.44 setBroadcastToAddressBlockRef

Description: Sets the addressBlock for a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to a broadcastTo element
- Input: addressBlockRef of type **String** - Name of the referenced addressBlock

F.7.74.45 setBroadcastToAlternateRegisterRef

Description: Sets an alternateRegisterRef on a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle of a broadcastTo element
- Input: alternateRegisterRef of type **String** - the alternateRegisterRef value to set

F.7.74.46 setBroadcastToFieldRef

Description: Sets the fieldRef for a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to a broadcastTo element
- Input: fieldRef of type **String** - Handle to a field to a register

F.7.74.47 setBroadcastToMemoryMapRef

Description: Sets the memoryMapRef for a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to a broadcastTo element
- Input: memoryMapRef of type **String** - Name of the referenced memoryMap

F.7.74.48 setBroadcastToRegisterRef

Description: Sets the RegisterRef for a broadcastTo element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: broadcastToID of type **String** - Handle to a broadcastTo element
- Input: registerRef of type **String** - Name of the referenced register

F.7.74.49 setEnumeratedValueUsage

Description: Sets the value of the usage field of an enumeratedValue

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: enumeratedValueID of type **String** - Handle to an enumeratedValue element
- Input: usage of type **String** - Usage enumerated value. Can be one of: read, write or read-write

F.7.74.50 setEnumeratedValueValue

Description: Sets the expression associated with the given EnumeratedValue.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: enumeratedValueID of type **String** - Handle to an enumeratedValue element
- Input: expression of type **String** - New expression

F.7.74.51 setEnumeratedValuesEnumerationDefinitionRef

Description: Sets the enumerationDefinitionRef on the given enumeratedValues element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: enumeratedValuesID of type **String** - Habndle to an enumeratedValues element
- Input: enumerationDefinitionRef of type **String** - The enumerationDefinition reference
- Input: typeDefinitions of type **String** - The referenced externalType definition

F.7.74.52 setFieldDefinitionBitWidth

Description: Sets the bitWidth expression of the given fieldDefinition.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldDefinitionID of type **String** - Handle to the fieldDefinition
- Input: value of type **String** - Handle to the new value

F.7.74.53 setFieldDefinitionTypeIdentifier

Description: Sets the typeIdentifier string of the given fieldDefinition.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldDefinitionID of type **String** - Handle to the fieldDefinition
- Input: value of type **String** - Handle to the new value

F.7.74.54 setFieldDefinitionVolatile

Description: Sets the volatile Boolean of the given fieldDefinition.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldDefinitionID of type **String** - the identifier of the fieldDefinition
- Input: value of type **Boolean** - Handle to the new value

F.7.74.55 setModeRefPriority

Description: Sets the priority on the given modeRef.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeRefID of type **String** - Handle to a modeRef element
- Input: priority of type **Long** - Priority value

F.7.74.56 setRegisterAddressOffset

Description: Sets the addressOffset for the given register element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerID of type **String** - Handle to a register element
- Input: value of type **String** - value to set on the addressOffset

F.7.74.57 setRegisterFieldAliasOf

Description: Sets an aliasOf element on a field of a register and returns his ID.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFieldID of type **String** - Handle to an field on a register element
- Input: fieldRef of type **String** - Name of the referenced field

F.7.74.58 setRegisterFieldBitWidth

Description: Sets the bitWith on a field of a register element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFieldID of type **String** - Handle to a register field element
- Input: value of type **String** - Handle value of the bitWidth

F.7.74.59 setRegisterFieldEnumeratedValues

Description: Sets the enumeratedValues element on the given field of register element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFieldID of type **String** - Handle to the field on a register

F.7.74.60 setRegisterFieldFieldAccessPolicies

Description: Sets the fieldACcessPolicies on the given field on a register.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFieldID of type **String** - Handle to an field on a register element

F.7.74.61 setRegisterFieldFieldDefinitionRef

Description: Sets the fieldDefinitionRef on the given field of a register.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFieldID of type **String** - Handle to a field element on a register
- Input: value of type **String** - Handle to th fieldDefinitionRef value
- Input: typeDefinitions of type **String** - the typeDefinitions value

F.7.74.62 setRegisterFieldTypeIdentifier

Description: Sets the typeIdentifier with the given value to the given register field element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: regFieldID of type **String** - Handle to a register field element
- Input: typeIdentifier of type **String** - Register field typeIdentifier

F.7.74.63 setRegisterFieldVolatility

Description: Sets the volatile flag with the given value to the given register field element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: `regFieldID` of type ***String*** - Handle to a register field element
- Input: `volatile` of type ***Boolean*** - Register field volatility

F.7.74.64 setRegisterRegisterDefinitionRef

Description: Sets the registerDefinitionRef on the register.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `value` of type ***String*** - Name of the referenced registerDefinition in an `externalTypeDefinitions`
- Input: `typeDefinitions` of type ***String*** - Name of the component `externalTypeDefinitions`

F.7.74.65 setRegisterSize

Description: Sets the size for the given register element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `value` of type ***String*** - Value to set on the size

F.7.74.66 setRegisterTypeIdentifier

Description: Sets typeIdentifier with the given value for the given register element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `typeIdentifier` of type ***String*** - Register typeIdentifier

F.7.74.67 setRegisterVolatility

Description: Sets volatility with the given value for the given register element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `volatile` of type ***Boolean*** - Register volatility

F.7.74.68 setResetMask

Description: Sets the mask with the given value for the given reset element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `resetID` of type ***String*** - Handle to a reset element
- Input: `mask` of type ***String*** - Reset mask expression

F.7.74.69 setResetValue

Description: Sets the value on a reset element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `resetID` of type ***String*** - Handle to a reset element
- Input: `value` of type ***String*** - Value to set on the reset

F.7.75 Register file (BASE)

F.7.75.1 getAliasOfRegisterFileRefByNames

Description: Returns all the registerFileRefs defined on the given aliasOf element.

- Returns: registerFileRef of type **String** - List of registerFileRef names
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.75.2 getAliasOfRegisterFileRefIDs

Description: Returns the handles to all the registerFileRefs defined on the given aliasOf element.

- Returns: registerFileRefIDs of type **String** - List of handles to the referenced registerFile elements
- Input: aliasOfID of type **String** - Handle to an aliasOf element

F.7.75.3 getRegisterFileAccessHandleIDs

Description: Returns the handles to all the accessHandles defined on the given registerFile element.

- Returns: accessHandleIDs of type **String** - List of handles to accessHandle elements
- Input: registerFileID of type **String** - Handle to a registerFile element

F.7.75.4 getRegisterFileAccessPolicyIDs

Description: Returns the handles to all the accessPolicies defined on the given registerFile element.

- Returns: accessPolicyIDs of type **String** - List of handles to the accessPolicy elements
- Input: registerFileID of type **String** - Handle to a registerFile element

F.7.75.5 getRegisterFileAddressOffset

Description: Returns the addressOffset in the given registerFile element.

- Returns: addressOffset of type **Long** - RegisterFile addressOffset
- Input: registerFileID of type **String** - Handle to a registerFile element

F.7.75.6 getRegisterFileAddressOffsetExpression

Description: Returns the addressOffset expression defined on the given registerFile element.

- Returns: addressOffset of type **String** - The addressOffset expression
- Input: registerFileID of type **String** - Handle to a registerFile element

F.7.75.7 getRegisterFileArrayID

Description: Returns the handle to the array defined on the given registerFile element.

- Returns: arrayID of type **String** - Handle to the registerFile array
- Input: registerFileID of type **String** - Handle to a registerFile element

F.7.75.8 getRegisterFileRange

Description: Returns the range defined on the given registerFile element.

- Returns: range of type **Long** - The range value (expressed as the number of addressable units)
- Input: registerFileID of type **String** - Handle to a registerFile element

F.7.75.9 getRegisterFileRangeExpression

Description: Returns the range expression defined on the given registerFile element.

- Returns: `range` of type ***String*** - The range expression
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.75.10 getRegisterFileRangeID

Description: Returns the handle to the range defined on the given registerFile element.

- Returns: `rangeID` of type ***String*** - Handle to the range
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.75.11 getRegisterFileRefIndexIDs

Description: Returns the handles to all the indices defined on the given addressBlockRef element.

- Returns: `indicesIDs` of type ***String*** - List of handles to the index elements
- Input: `addressBlockRefID` of type ***String*** - Handle to an addressBlockRef element

F.7.75.12 getRegisterFileRegisterFileDefinitionRefByExternalTypeDefID

Description: Returns the handle to the externalTypeDefinitions referenced by the typeDefinitions attribute of the registerFileDefinitionRef defined on the given registerFile element.

- Returns: `registerFileDefinitionID` of type ***String*** - Handle to the externalTypeDefinitions element referenced by the typeDefinitions attribute
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.75.13 getRegisterFileRegisterFileDefinitionRefByID

Description: Returns the handle to the registerFileDefinition referenced from the given registerFile element.

- Returns: `registerFileDefinitionID` of type ***String*** - Handle to the referenced registerFileDefinition element
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.75.14 getRegisterFileRegisterFileDefinitionRefByName

Description: Returns the registerFileDefinitionRef value defined on the given registerFile element.

- Returns: `registerFileDefinitionRef` of type ***String*** - The registerFileDefinitionRef on a registerFile element
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.75.15 getRegisterFileRegisterFileDefinitionRefID

Description: Returns the handle to the registerFileDefinitionRef defined on the given registerFile element.

- Returns: `registerFileDefinitionRefID` of type ***String*** - Handle to the registerFileDefinitionRef element
- Input: `registerFileID` of type ***String*** - Handle to the a registerFile element

F.7.75.16 getRegisterFileRegisterFileIDs

Description: Returns the handles to all the registerFiles defined on the given registerFile element.

- Returns: `registerFileIDs` of type ***String*** - List of handles to registerFile elements
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.75.17 getRegisterFileRegisterIDs

Description: Returns the handles to all the registers defined on the given registerFile element.

- Returns: `registerIDs` of type ***String*** - List of handles to the register elements
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.75.18 getRegisterFileTypeIdentifier

Description: Returns the typeIdentifier defined on the given registerFile element.

- Returns: `typeIdentifier` of type ***String*** - The typeIdentifier value
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.76 Register file (EXTENDED)

F.7.76.1 addAliasOfRegisterFileRef

Description: Adds a registerFileRef on an aliasOf of a field of a register element.

- Returns: `registerFileRefID` of type ***String*** - the registerFileRef identifier on the aliasOf
- Input: `aliasOfID` of type ***String*** - Handle to an aliasOf element
- Input: `registerFileRef` of type ***String*** - Name of the referenced registerFile

F.7.76.2 addRegisterFileDefinitionRegisterFile

Description: Adds a registerFile to the given registerFileDefinition element.

- Returns: `registerFileID` of type ***String*** - Handle to the added registerFile
- Input: `registerFileDefinitionID` of type ***String*** - Handle to a registerFileDefinition
- Input: `name` of type ***String*** - Handle to the name of the registerFile to create
- Input: `addressOffset` of type ***String*** - Handle to a registerFile address offset
- Input: `range` of type ***String*** - Handle to a registerFile range

F.7.76.3 addRegisterFileRefIndex

Description: Adds an index to a registerFileRef element.

- Returns: `indexID` of type ***String*** - Handle to the added index
- Input: `registerFileRefID` of type ***String*** - Handle to a registerFileRef element
- Input: `value` of type ***String*** - Index value

F.7.76.4 addRegisterFileRegister

Description: Adds a register with the given name, offset, and size, and a field with the given name, offset, and width to the given registerFile.

- Returns: `registerID` of type ***String*** - Handle to a new register element
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element
- Input: `name` of type ***String*** - Register name
- Input: `offset` of type ***String*** - Register offset

- Input: `size` of type ***String*** - Register size
- Input: `fieldName` of type ***String*** - Field name
- Input: `fieldOffset` of type ***String*** - Field offset
- Input: `fieldWidth` of type ***String*** - Field width

F.7.76.5 removeAliasOfRegisterFileRef

Description: Removes the given registerFileRef from its containing aliasOf element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileRefID` of type ***String*** - Handle to a registerFileRef element

F.7.76.6 removeRegisterFileAccessHandle

Description: Removes the given access handle from its containing register file element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessHandleID` of type ***String*** - Handle to the accessHandle to remove

F.7.76.7 removeRegisterFileRefIndex

Description: Removes the given index from its containing registerFileRef element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indexID` of type ***String*** - Handle to an index element

F.7.76.8 removeRegisterFileRegister

Description: Removes the given register from its containing registerFile element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.76.9 removeRegisterFileRegisterFile

Description: Removes the given RegisterFile from its containing registerFile element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileID` of type ***String*** - Handle to the element to remove

F.7.76.10 removeRegisterFileTypeIdentifier

Description: Removes the typeIdentifier with the given value in the given registerFile element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.76.11 setRegisterFileAddressOffset

Description: Sets the value of the addressOffset expression of the given registerFile.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileID` of type ***String*** - Handle to a registerFile
- Input: `expression` of type ***String*** - the new value for the expression

F.7.76.12 setRegisterFileRange

Description: sets the range expression of a registerFile of the given registerFile.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileID of type **String** - Handle to a registerFile
- Input: expression of type **String** - Handle to the new value for the expression

F.7.76.13 setRegisterFileRegisterFileDefinitionRef

Description: Sets the registerFileDefinitionRef on a registerFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileID of type **String** - Handle to a registerFile element
- Input: value of type **String** - Name of the referenced registerFileDefinition in an externalTypeDefinitions
- Input: typeDefinitions of type **String** - Name of the component externalTypeDefinitions

F.7.76.14 setRegisterFileTypeIdentifier

Description: Sets the typeIdentifier with the given value in the given registerFile element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileID of type **String** - Handle to a registerFile element
- Input: typeIdentifier of type **String** - RegisterFile typeidentifier

F.7.77 Slice (BASE)

F.7.77.1 getFieldSliceAddressBlockRefByID

Description: Returns the addressBlockID referenced by the given fieldSlice element.

- Returns: addressBlockID of type **String** - Handle to the referenced addressBlock element
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element

F.7.77.2 getFieldSliceAddressBlockRefByName

Description: Returns the addressBlockRef defined on the given fieldSlice element.

- Returns: addressBlockRef of type **String** - The referenced addressBlock
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element

F.7.77.3 getFieldSliceAddressBlockRefID

Description: Returns the handle to the addressBlockRef defined on the given fieldSlice element.

- Returns: addressBlockRefID of type **String** - Handle to the addressBlockRef element
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element

F.7.77.4 getFieldSliceAddressSpaceRefByID

Description: Returns the addressSpaceID from the given fieldSlice element.

- Returns: addressSpaceID of type **String** - Handle to a addressSpace element
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element

F.7.77.5 getFieldSliceAddressSpaceRefByName

Description: Returns the addressSpaceRef defined on the given fieldSlice element.

- Returns: addressSpaceRef of type ***String*** - The referenced addressSpace
- Input: fieldSliceID of type ***String*** - Handle to a fieldSlice element

F.7.77.6 getFieldSliceAddressSpaceRefID

Description: Returns the handle to the addressSpaceRef defined on the given fieldSlice element.

- Returns: addressSpaceRefID of type ***String*** - Handle to the addressSpaceRef element
- Input: fieldSliceID of type ***String*** - Handle to a fieldSlice element

F.7.77.7 getFieldSliceAlternateRegisterRefByID

Description: Returns the alternateRegisterID from the given fieldSlice element.

- Returns: alternateRegisterID of type ***String*** - Handle to a alternateRegister element
- Input: fieldSliceID of type ***String*** - Handle to a fieldSlice element

F.7.77.8 getFieldSliceAlternateRegisterRefByName

Description: Returns the alternateRegisterRef defined on the given fieldSlice element.

- Returns: alternateRegisterRef of type ***String*** - The referenced alternateRegister name
- Input: fieldSliceID of type ***String*** - Handle to a fieldSlice element

F.7.77.9 getFieldSliceAlternateRegisterRefID

Description: Returns the handle to the alternateRegisterRef defined on the given fieldSlice element.

- Returns: alternateRegisterRefID of type ***String*** - Handle to the alternateRegisterRef element
- Input: fieldSliceID of type ***String*** - Handle to a fieldSlice element

F.7.77.10 getFieldSliceBankRefByNames

Description: Returns all the bankRefs defined on the given fieldSlice element.

- Returns: bankRefs of type ***String*** - List of the referenced banks
- Input: fieldSliceID of type ***String*** - Handle to a fieldSlice element

F.7.77.11 getFieldSliceBankRefIDs

Description: Returns the handles to all the bankRefs defined on the given fieldSlice element

- Returns: bankRefIDs of type ***String*** - List of handles to the bankRef elements
- Input: fieldSliceID of type ***String*** - Handle to a fieldSlice

F.7.77.12 getFieldSliceFieldRefByName

Description: Returns the FieldRef defined on the given fieldSlice element.

- Returns: FieldRef of type ***String*** - The referenced field element
- Input: fieldSliceID of type ***String*** - Handle to a fieldSlice

F.7.77.13 getFieldSliceFieldRefID

Description: Returns the handle to the fieldRef defined on the given fieldSlice element.

- Returns: `FieldRefID` of type ***String*** - Handle to the fieldRef element
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.14 getFieldSliceMemoryMapRefByID

Description: Returns the memoryMapID referenced by the given fieldSlice element.

- Returns: `memoryMapID` of type ***String*** - Handle to the referenced memoryMap element
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.15 getFieldSliceMemoryMapRefByName

Description: Returns the memoryMapRef defined on the given fieldSlice element.

- Returns: `memoryMapRef` of type ***String*** - The referenced memoryMap
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.16 getFieldSliceMemoryMapRefID

Description: Returns the handle to the memoryMapRef defined on the given fieldSlice element.

- Returns: `memoryMapRefID` of type ***String*** - Handle to the referenced memoryMap
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.17 getFieldSliceMemoryRemapRefByID

Description: Returns the memoryRemapID referenced by the given fieldSlice element.

- Returns: `memoryRemapID` of type ***String*** - Handle to the referenced memoryRemap element
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.18 getFieldSliceMemoryRemapRefByName

Description: Returns the memoryRemapRef defined on the given fieldSlice element.

- Returns: `memoryRemapRef` of type ***String*** - The memoryMap reference
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.19 getFieldSliceMemoryRemapRefID

Description: Returns the handle to the memoryRemapRef defined on the given fieldSlice element.

- Returns: `memoryRemapRefID` of type ***String*** - Handle to the memoryMap reference
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.20 getFieldSliceRange

Description: Returns the range defined on the given fieldSlice element.

- Returns: `values` of type ***String*** - Array of two range values: left and right
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.21 getFieldSliceRangeLeftID

Description: Returns the handle to the left range defined on the given fieldSlice element.

- Returns: `leftID` of type ***String*** - Handle to the left element
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.22 getFieldSliceRangeRightID

Description: Returns the handle to the right range defined on the given fieldSlice element.

- Returns: `rightID` of type ***String*** - Handle to the right element
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.23 getFieldSliceRegisterFileRefByNames

Description: Returns all the registerFileRefs defined on the given fieldSlice element.

- Returns: `registerFileRef` of type ***String*** - List of the referenced registerFiles
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.24 getFieldSliceRegisterFileRefs

Description: Returns the handles to all the registerFileRefs defined on the given fieldSlice element.

- Returns: `registerFileID` of type ***String*** - List of the registerFileRef elements
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.25 getFieldSliceRegisterRefByID

Description: Returns the registerID referenced by the given fieldSlice element.

- Returns: `registerID` of type ***String*** - Handle to the referenced register element
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.26 getFieldSliceRegisterRefByName

Description: Returns the registerRef defined on the given fieldSlice element.

- Returns: `registerRef` of type ***String*** - The referenced registerRef element
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.27 getFieldSliceRegisterRefID

Description: Returns the handle to the registerRef defined on the given fieldSlice element.

- Returns: `registerRefID` of type ***String*** - Handle to the registerRef element
- Input: `fieldSliceID` of type ***String*** - Handle to a fieldSlice element

F.7.77.28 getLocationSliceIDs

Description: Returns the handles to all the slices defined on the given location element.

- Returns: `sliceIDs` of type ***String*** - List of handles to the slice elements
- Input: `locationID` of type ***String*** - Handle to a location element

F.7.77.29 **getModeFieldSliceIDs**

Description: Returns all the handle to the fieldSlice defined on the given mode element.

- Returns: `fieldSliceIDs` of type ***String*** - List of handles to the fieldSlice elements
- Input: `modeID` of type ***String*** - Handle to a mode element

F.7.77.30 **getModePortSliceIDs**

Description: Returns the handles to all the slices defined on the given mode element.

- Returns: `portSliceIDs` of type ***String*** - List of handles to the portSlice elements
- Input: `modeID` of type ***String*** - Handle to a mode element

F.7.77.31 **getPortSlicePartSelectID**

Description: Returns the handle to the partSelect defined on the given portSlice element.

- Returns: `partSelectID` of type ***String*** - Handle to the partSelect element
- Input: `portSliceID` of type ***String*** - Handle to a portSlice element

F.7.77.32 **getPortSlicePortRefByName**

Description: Returns the portRef defined on the given portSlice element.

- Returns: `portRef` of type ***String*** - The referenced port
- Input: `portSliceID` of type ***String*** - Handle to a portSlice element

F.7.77.33 **getPortSliceSubPortReferenceIDs**

Description: Returns all the subPortRefs defined on a portSlice element.

- Returns: `subPortRefs` of type ***String*** - List of the referenced subPorts
- Input: `portSliceID` of type ***String*** - Handle to a portSlice element

F.7.77.34 **getSlicePathSegmentIDs**

Description: Returns the handles to all the pathSegments defined on the given slice element.

- Returns: `pathSegmentIDs` of type ***String*** - List of handles to pathSegment elements
- Input: `sliceID` of type ***String*** - Handle to a slice element

F.7.77.35 **getSliceRange**

Description: Returns the range defined on the given slice element.

- Returns: `range` of type ***Long*** - Array of two range values: left and right
- Input: `sliceID` of type ***String*** - Handle to a slice element

F.7.77.36 **getSliceRangeExpression**

Description: Returns the range left and right expressions defined on the given slice element.

- Returns: `expressions` of type ***String*** - Array of two range expressions: left and right
- Input: `sliceID` of type ***String*** - Handle to a slice element

F.7.77.37 **getSliceRangeLeftID**

Description: Returns the handle to the left range defined on the given slice element.

- Returns: `leftRangeID` of type **String** - Handle to the left range
- Input: `sliceID` of type **String** - Handle to a slice element

F.7.77.38 **getSliceRangeRightID**

Description: Returns the handle to the right range defined on the given slice element.

- Returns: `rightRangeID` of type **String** - Handle to the right range
- Input: `sliceID` of type **String** - Handle to a slice element

F.7.78 Slice (EXTENDED)

F.7.78.1 **addFieldSliceBankRef**

Description: Adds bank reference to a fieldSlice element.

- Returns: `bankRefID` of type **String** - Handle to the new bankRef element
- Input: `fieldSliceID` of type **String** - Handle to a fieldSlice element
- Input: `bankRef` of type **String** - Name of the referenced bank

F.7.78.2 **addFieldSliceRegisterFileRef**

Description: Adds a registerFileRef on a fieldSlice element.

- Returns: `registerFileRefID` of type **String** - the registerFileRef identifier
- Input: `fieldSliceID` of type **String** - Handle to a fieldSlice element
- Input: `registerFileRef` of type **String** - Name of the referenced registerFile

F.7.78.3 **addModeFieldSlice**

Description: Adds an fieldSlice on mode element.

- Returns: `fieldSliceID` of type **String** - a fieldSlice identifier
- Input: `modeID` of type **String** - Handle to a mode element
- Input: `name` of type **String** - Name of a fieldSlice
- Input: `memoryMapRef` of type **String** - Name of the referenced memoryMap
- Input: `addressBlockRef` of type **String** - Name of the referenced addressBlock
- Input: `registerRef` of type **String** - Name of the referenced register
- Input: `fieldRef` of type **String** - Name of the referenced field

F.7.78.4 **addModePortSlice**

Description: Adds a portSlice on a Mode element.

- Returns: `portSliceID` of type **String** - Handle to the added portSlice
- Input: `modeID` of type **String** - Handle to a mode element
- Input: `name` of type **String** - Name of the port slice
- Input: `portRef` of type **String** - Name of the referenced port

F.7.78.5 addPortSliceSubPortReference

Description: Adds a subPortReference to an portSlice element.

- Returns: subPortReferenceID of type **String** - Handle to the added subPortReference
- Input: portSliceID of type **String** - Handle to a portSlice element
- Input: subPortRef of type **String** - Name of the referenced subPort

F.7.78.6 addSlicePathSegment

Description: Adds a pathSegment element to a given slice element.

- Returns: pathSegmentID of type **String** - Handle to the added pathSegment
- Input: sliceID of type **String** - Handle to a slice element
- Input: pathSegmentValue of type **String** - Handle to value of the pathSegment

F.7.78.7 removeFieldSliceAlternateRegisterRef

Description: Removes an alternateRegisterRef on a fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle of a fieldSlice element

F.7.78.8 removeFieldSliceBankRef

Description: Removes the given bank reference from the given fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankRefID of type **String** - Handle to a bankRef element

F.7.78.9 removeFieldSliceMemoryRemapRef

Description: Removes the memoryRemap reference on an fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to an fieldSlice element

F.7.78.10 removeFieldSliceRange

Description: Removes a range on the given fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element

F.7.78.11 removeFieldSliceRegisterFileRef

Description: Removes the given registerFileRef from its containing fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element

F.7.78.12 removeLocationSlice

Description: Removes the given slice from its containing location element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: sliceID of type **String** - Handle to a slice element

F.7.78.13 removeModeFieldSlice

Description: Removes the given fieldSlice from its containing mode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element

F.7.78.14 removeModePortSlice

Description: Removes the given portSlice from its containing mode element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portSliceID of type **String** - Handle to a portSlice element

F.7.78.15 removePortSlicePartSelect

Description: Removes a partSelect on the given portSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portSliceID of type **String** - Handle to a portMap portSlice element

F.7.78.16 removePortSliceSubPortReference

Description: Removes the given subPortReference from its containing portSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: subPortReferenceID of type **String** - Handle to a subPortReference element

F.7.78.17 removeSlicePathSegment

Description: Removes the given pathSegment element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: pathSegmentID of type **String** - Handle to a pathSegment element

F.7.78.18 removeSliceRange

Description: Removes the range from the given slice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: sliceID of type **String** - Handle to a slice element

F.7.78.19 setFieldSliceAddressBlockRef

Description: Sets the addressBlockRef of a fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element
- Input: addressBlockRef of type **String** - Name of the referenced addressBlock

F.7.78.20 setFieldSliceAddressSpaceRef

Description: Sets the addressSpaceRef set on a fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element
- Input: addressSpaceRef of type **String** - set the addressSpaceRef on a fieldSlice element

F.7.78.21 setFieldSliceAlternateRegisterRef

Description: Sets an alternateRegisterRef on a fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle of a fieldSlice element
- Input: alternateRegisterRef of type **String** - the alternateRegisterRef value to set

F.7.78.22 setFieldSliceFieldRef

Description: Sets the fieldRef on a fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element
- Input: fieldRef of type **String** - Name of the referenced field

F.7.78.23 setFieldSliceMemoryMapRef

Description: Sets the memoryMapRef on a fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to an fieldSlice element
- Input: memoryMapRef of type **String** - Name of the referenced memoryMap

F.7.78.24 setFieldSliceMemoryRemapRef

Description: Sets the memoryRemapRef on a fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to an fieldSlice element
- Input: memoryRemapRef of type **String** - Name of the referenced memoryRemap

F.7.78.25 setFieldSliceRange

Description: Sets a range on the given fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element
- Input: range of type **String[]** - Range, defined as a pair of left and right value expressions

F.7.78.26 setFieldSliceRegisterRef

Description: Sets the registerRef of a fieldSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldSliceID of type **String** - Handle to a fieldSlice element
- Input: registerRef of type **String** - Name of the referenced register

F.7.78.27 setPortSlicePartSelect

Description: Set a partSelect on the given portSlice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: portSliceID of type **String** - Handle to a portMap portSlice element

- Input: `range` of type ***String[]*** - Create the range on the partSelect with “left” for range[0] and “right” for range[1]. Set to null if you only want indices.
- Input: `indices` of type ***String[]*** - Handle to values of type String. Set all the index on the partSelect

F.7.78.28 setPortSlicePortRef

Description: Sets the portRef on a portSlice element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portSliceID` of type ***String*** - Handle to a portSlice element
- Input: `portRef` of type ***String*** - Name of the referenced port

F.7.78.29 setSliceRange

Description: Sets the range to the given range for the given slice element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `sliceID` of type ***String*** - Handle to a slice element
- Input: `range` of type ***String[]*** - Range with left at index 0 and right at index 1

F.7.79 Top element (BASE)

F.7.79.1 getAbstractionDefIDs

Description: Returns the handles to all the registered abstractionDefinition objects.

- Returns: `abstractionDefIDs` of type ***String*** - List of handles to abstractionDefinition objects

F.7.79.2 getAbstractorIDs

Description: Returns the handles to all the registered abstractor objects.

- Returns: `abstractorIDs` of type ***String*** - List of handles to abstractor objects

F.7.79.3 getBusDefIDs

Description: Returns the handles to all the registered busDefinition objects.

- Returns: `busDefIDs` of type ***String*** - List of handles to busDefinition objects

F.7.79.4 getCatalogIDs

Description: Returns the handles to all the registered catalog objects.

- Returns: `catalogIDs` of type ***String*** - List of handles to catalog objects

F.7.79.5 getComponentIDs

Description: Returns the handles to all the registered component objects.

- Returns: `componentIDs` of type ***String*** - List of handles to component objects

F.7.79.6 getDesignConfigurationIDs

Description: Returns the handles to all the registered designConfiguration objects.

- Returns: designConfigurationIDs of type **String** - List of handles to designConfiguration objects

F.7.79.7 getDesignIDs

Description: Returns the handles to all the registered design objects.

- Returns: designIDs of type **String** - List of handles to design objects

F.7.79.8 getGeneratorChainIDs

Description: Returns the handles to all the registered generatorChain objects.

- Returns: generatorChainIDs of type **String** - List of handles to generatorChain objects

F.7.79.9 getID

Description: Returns the handle to the object correspondintg to the given VLNV.

- Returns: rootObjectID of type **String** - Handle to the object with the given VLNV
- Input: VLNV of type **String[]** - The VLNV of the root object

F.7.79.10 getTypeDefinitionsIDs

Description: Returns the handles to all the registered typeDefinitions objects.

- Returns: typeDefinitionsIDs of type **String** - List of handles to typeDefinitions objects

F.7.79.11 getVLNV

Description: Returns the VLNV of the object with the given ID.

- Returns: VLNV of type **String** - The VLNV of the object with the given ID
- Input: rootObjectID of type **String** - Handle to a root object

F.7.79.12 getXMLPath

Description: Returns the location of the XML file that is registered with the VLNV of the given object.

- Returns: path of type **String** - The location of the XML file
- Input: rootObjectID of type **String** - Handle to a root object
- Input: dereference of type **Boolean** - True if the links have to be dereferenced

F.7.80 Top element (EXTENDED)

F.7.80.1 createAbstractionDef

Description: Creates a new abstractionDef with the given VLNV, busDefVLNV, logicalName and type and returns its abstractionDefID; fails and returns null if VLNV already exists.

- Returns: abstractionDefID of type **String** - Handle to a new abstractionDef element
- Input: abstractionDefVLNV of type **String[]** - abstractionDef VLNV
- Input: busDefVLNV of type **String[]** - busDef VLNV
- Input: logicalName of type **String** - Logical port name
- Input: type of type **String** - Logical port style (wire or transactional)

F.7.80.2 createAbstractor

Description: Creates a new abstractor with the given VLVN and returns its abstractorID; fails and returns null if VLVN already exists.

- Returns: abstractorID of type **String** - Handle to a new abstractor
- Input: vlnv of type **String[]** - VLVN for a new abstractor
- Input: mode of type **String** - Abstractor mode
- Input: busDefVLNV of type **String[]** - busDef VLVN
- Input: firstBusInterfaceName of type **String** - First busInterface name
- Input: secondBusInterfaceName of type **String** - Second busInterfaceName

F.7.80.3 createBusDefinition

Description: Creates a new busDef with the given VLVN, directConnection, and isAddressable and returns its busDefID; fails and returns null if VLVN already exists.

- Returns: busDefID of type **String** - Handle to a new busDef element
- Input: vlnv of type **String[]** - busDef VLVN
- Input: directConnection of type **Boolean** - busDef directionConnection
- Input: isAddressable of type **Boolean** - busDef isAddressable

F.7.80.4 createCatalog

Description: Creates a new catalog and returns its catalogID; fails and returns null if VLVN already exists.

- Returns: catalogID of type **String** - Handle to a new catalog element
- Input: vlnv of type **String[]** - Catalog VLVN

F.7.80.5 createComponent

Description: Creates a new component with the given VLVN and returns its componentID; fails and returns null if VLVN already exists.

- Returns: componentID of type **String** - Handle to a new component
- Input: vlnv of type **String[]** - VLVN for a new component

F.7.80.6 createDesign

Description: Creates a new design with the given VLVN and returns its designID; fails and returns null if VLVN already exists.

- Returns: designID of type **String** - Handle to a design element
- Input: vlnv of type **String[]** - Design VLVN

F.7.80.7 createDesignConfiguration

Description: Creates a new designConfiguration with the given VLVN and returns its designID; fails and returns null if VLVN already exists.

- Returns: designConfigurationID of type **String** - Handle to a designConfiguration element
- Input: vlnv of type **String[]** - designConfiguration VLVN

F.7.80.8 createGeneratorChain

Description: Creates a new generatorChain with the given VLVN and returns its generatorChainID; fails and returns null if VLVN already exists.

- Returns: generatorChainID of type **String** - Handle to a new generatorChain element
- Input: generatorChainVLNV of type **String[]** - generatorChain VLVN
- Input: generatorName of type **String** - The name value to be set on the generator
- Input: generatorExe of type **String** - The generatorExe value to be set

F.7.80.9 createTypeDefinitions

Description: Creates a new typeDefinitions with the given VLVN and returns its typeDefinitionsID; fails and returns null if VLVN already exists.

- Returns: typeDefinitionsID of type **String** - Handle to the added typeDefinitions
- Input: typeDefinitionsVLNV of type **String[]** - VLVN for a new typeDefinitions

F.7.80.10 edit

Description: Signal start of editing of the given top-element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: rootObjectID of type **String** - Handle to a top-level element

F.7.80.11 setXMLPath

Description: Sets new location for the XML file that is registered with the VLVN of the given top-element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: rootObjectID of type **String** - Handle to a top-level element
- Input: path of type **String** - New location of XML file

F.7.81 Type definitions (BASE)

F.7.81.1 getAddressBlockDefinitionAddressUnitBits

Description: Returns the addressUnitBits (resolved) value defined on the given addressBlockDefinition element.

- Returns: addressUnitBits of type **Long** - The addressUnitBits value
- Input: addressBlockDefinitionID of type **String** - Handle to an addressBlockDefinition element

F.7.81.2 getAddressBlockDefinitionAddressUnitBitsExpression

Description: Returns the addressUnitBits expression defined on the given addressBlockDefinition element.

- Returns: addressUnitBits of type **String** - The addressUnitBits expression
- Input: addressBlockDefinitionID of type **String** - Handle to an addressBlockDefinition element

F.7.81.3 getAddressBlockDefinitionAddressUnitBitsID

Description: Returns the handle to the addressUnitBits defined on the given addressBlockDefinition element.

- Returns: addressUnitBitsID of type **String** - Handle to the addressUnitBits element
- Input: addressBlockDefinitionID of type **String** - Handle to an addressBlockDefinition element

F.7.81.4 getBankDefinitionAddressUnitBits

Description: Returns the addressUnitBits (resolved) value defined on the given bankDefinition element.

- Returns: addressUnitBits of type **Long** - The addressUnitBits value
- Input: bankDefinitionID of type **String** - Handle to an bankDefinition element

F.7.81.5 getBankDefinitionAddressUnitBitsExpression

Description: Returns the addressUnitBits expression defined on the given bankDefinition element.

- Returns: addressUnitBits of type **String** - The addressUnitBits expression
- Input: bankDefinitionID of type **String** - Handle to an bankDefinition element

F.7.81.6 getBankDefinitionAddressUnitBitsID

Description: Returns the handle to the addressUnitBits defined on the given bankDefinition element.

- Returns: addressUnitBitsID of type **String** - Handle to the addressUnitBits element
- Input: bankDefinitionID of type **String** - Handle to an bankDefinition element

F.7.81.7 getEnumerationDefinitionEnumeratedValueIDs

Description: Returns the handles of all the enumeratedValues defined on the given enumerationDefinition element.

- Returns: enumeratedValueIDs of type **String** - List of handles to the enumeratedValue elements
- Input: enumerationDefinitionID of type **String** - Handle to an enumerationDefinition element

F.7.81.8 getEnumerationDefinitionWidth

Description: Returns the (resolved) width value defined on the given enumerationDefinition element.

- Returns: width of type **Long** - The value of the width element
- Input: enumerationDefinitionID of type **String** - Handle to an enumerationDefinition element

F.7.81.9 getEnumerationDefinitionWidthExpression

Description: Returns the width expression defined on the given enumerationDefinition element.

- Returns: width of type **String** - The width expression
- Input: enumerationDefinitionID of type **String** - Handle to an enumerationDefinition element

F.7.81.10 getEnumerationDefinitionWidthID

Description: Returns the handle to the width element defined on the given enumerationDefinition element.

- Returns: widthID of type **String** - Handle to the width element
- Input: enumerationDefinitionID of type **String** - Handle to an enumerationDefinition element

F.7.81.11 getExternalTypeDefinitionsModeLinksIDs

Description: Returns the handles to all the modeLinks defined on the given externalTypeDefinitions element.

- Returns: modeLinksIDs of type **String** - List of handles to the modeLinks elements
- Input: externalTypeDefinitionsID of type **String** - Handle to an externalTypeDefinitions element

F.7.81.12 getExternalTypeDefinitionsResetTypeLinkIDs

Description: Returns the handles to all the resetTypeLinks defined on the given externalTypeDefinitions element.

- Returns: resetTypeLinksIDs of type **String** - List of handles to the resetTypeLinks elements
- Input: externalTypeDefinitionsID of type **String** - Handle to an externalTypeDefinitions element

F.7.81.13 getExternalTypeDefinitionsTypeDefinitionsRefByID

Description: Returns the handle to the typeDefinitions instance referenced from the given externalTypeDefinitions.

- Returns: typeDefinitionsID of type **String** - Handle to the referenced typeDefinitions object
- Input: externalTypeDefinitionsID of type **String** - Handle to an externalTypeDefinitions element

F.7.81.14 getExternalTypeDefinitionsTypeDefinitionsRefByVLNV

Description: Returns the VLNV of the typeDefinitions referenced from the given externalTypeDefinitions.

- Returns: VLNV of type **String** - The VLNV of the referenced typeDefinitions object
- Input: externalTypeDefinitionsID of type **String** - Handle to an externalTypeDefinitions element

F.7.81.15 getExternalTypeDefinitionsViewLinkIDs

Description: Returns the handles to all the viewLinks defined on the given externalTypeDefinitions element

- Returns: viewLinkIDs of type **String** - List of handles to the viewLinks elements
- Input: externalTypeDefinitionsID of type **String** - Handle to an externalTypeDefinitions element

F.7.81.16 getMemoryRemapDefinitionAddressUnitBits

Description: Returns the addressUnitBits (resolved) value defined on the given memoryRemapDefinition element.

- Returns: addressUnitBits of type **Long** - The addressUnitBits value

- Input: memoryRemapDefinitionID of type **String** - Handle to an memoryRemapDefinition element

F.7.81.17 getMemoryRemapDefinitionAddressUnitBitsExpression

Description: Returns the addressUnitBits expression defined on the given memoryRemapDefinition element.

- Returns: addressUnitBits of type **String** - The addressUnitBits expression
- Input: memoryRemapDefinitionID of type **String** - Handle to an memoryRemapDefinition element

F.7.81.18 getMemoryRemapDefinitionAddressUnitBitsID

Description: Returns the handle to the addressUnitBits defined on the given memoryRemapDefinition element.

- Returns: addressUnitBitsID of type **String** - Handle to the addressUnitBits element
- Input: memoryRemapDefinitionID of type **String** - Handle to an memoryRemapDefinition element

F.7.81.19 getModeLinkExternalModeReferenceRefByName

Description: Returns the externalModeReference defined on the given modeLink element.

- Returns: externalModeReference of type **String** - The referenced externalMode name
- Input: modeLinkID of type **String** - Handle to a modeLink element

F.7.81.20 getModeLinkExternalModeReferenceID

Description: Returns the handle to the externalModeReference defined on the given modeLink element.

- Returns: externalModeReferenceID of type **String** - Handle to the externalModeReference element
- Input: modeLinkID of type **String** - Handle to a modeLink element

F.7.81.21 getModeLinkModeReferenceRefByName

Description: Returns the modeReference defined on the given modeLink element.

- Returns: modeReference of type **String** - The referenced mode name
- Input: modeLinkID of type **String** - Handle to a modeLink element

F.7.81.22 getModeLinkModeReferenceID

Description: Returns the handle to the modeReference defined on the given modeLink element.

- Returns: modeReferenceID of type **String** - Handle to the modeReference element
- Input: modeLinkID of type **String** - Handle to a modeLink element

F.7.81.23 getRegisterFileDefinitionAddressUnitBits

Description: Returns the addressUnitBits (resolved) value defined on the given registerFileDefinition element.

- Returns: addressUnitBits of type **Long** - The addressUnitBits value

- Input: `registerFileDefinitionID` of type ***String*** - Handle to a `registerFileDefinition` element

F.7.81.24 getRegisterFileDefinitionAddressUnitBitsExpression

Description: Returns the `addressUnitBits` expression defined on the given `registerFileDefinition` element.

- Returns: `addressUnitBits` of type ***String*** - The `addressUnitBits` expression
- Input: `registerFileDefinitionID` of type ***String*** - Handle to a `registerFileDefinition` element

F.7.81.25 getRegisterFileDefinitionAddressUnitBitsID

Description: Returns the handle to the `addressUnitBits` defined on the given `registerFileDefinition` element.

- Returns: `addressUnitBitsID` of type ***String*** - Handle to the `addressUnitBits` element
- Input: `registerFileDefinitionID` of type ***String*** - Handle to a `registerFileDefinition` element

F.7.81.26 getResetTypeLinkExternalResetTypeRefByName

Description: Returns the `externalResetTypeRef` defined on the given `resetTypeLink` element.

- Returns: `externalResetTypeRef` of type ***String*** - The referenced `externalResetType`
- Input: `resetTypeLinkID` of type ***String*** - Handle to a `resetTypeLink` element

F.7.81.27 getResetTypeLinkExternalResetTypeReferenceID

Description: Returns the handle to the `externalResetTypeReference` defined on the given `resetTypeLink` element.

- Returns: `externalResetTypeReferenceID` of type ***String*** - Handle to the `externalResetTypeReference` element
- Input: `resetTypeLinkID` of type ***String*** - Handle to a `resetTypeLink` element

F.7.81.28 getResetTypeLinkResetTypeReferenceRefByName

Description: Returns the `resetTypeReference` defined on the given `resetTypeLink` element.

- Returns: `resetTypeReference` of type ***String*** - The referenced `resetType`
- Input: `resetTypeLinkID` of type ***String*** - Handle to a `resetTypeLink` element

F.7.81.29 getResetTypeLinkResetTypeReferenceID

Description: Returns the handle to the `resetTypeReference` defined on the given `resetTypeLink` element.

- Returns: `resetTypeReferenceID` of type ***String*** - Handle to the `resetTypeReference` element
- Input: `resetTypeLinkID` of type ***String*** - Handle to a `resetTypeLink` element

F.7.81.30 getTypeDefinitionsAddressBlockDefinitionIDs

Description: Returns the handles to all the `addressBlockDefinitions` defined on the given `typeDefinitions` element.

- Returns: `addressBlockDefinitionIDs` of type ***String*** - List of handles to the `addressBlockDefinition` elements
- Input: `typeDefinitionsID` of type ***String*** - Handle to a `typeDefinitions` element

F.7.81.31 getTypeDefinitionsBankDefinitionIDs

Description: Returns the handles to all the bankDefinitions defined on the given typeDefinitions element.

- Returns: bankDefinitionIDs of type ***String*** - List of handles to the bankDefinition elements
- Input: typeDefinitionsID of type ***String*** - Handle to a typeDefinitions element

F.7.81.32 getTypeDefinitionsChoiceIDs

Description: Returns the handles to all the choices defined on the given typeDefinitions element.

- Returns: choiceIDs of type ***String*** - List of handles to choice elements
- Input: typeDefinitionsID of type ***String*** - Handle to a typeDefinitions element

F.7.81.33 getTypeDefinitionsEnumerationDefinitionIDs

Description: Returns the handles to all the enumerationDefinitions defined on the given typeDefinitions element.

- Returns: enumerationDefinitionIDs of type ***String*** - List of handles to the enumerationDefinition elements
- Input: typeDefinitionsID of type ***String*** - Handle to a typeDefinitions element

F.7.81.34 getTypeDefinitionsExternalTypeDefinitionsIDs

Description: Returns the handles to all the externalTypeDefinitions defined on the given typeDefinitions element.

- Returns: externalTypeDefinitionsIDs of type ***String*** - List of handles to the externalTypeDefinitions elements
- Input: typeDefinitionsID of type ***String*** - Handle to a typeDefinitions element

F.7.81.35 getTypeDefinitionsFieldDefinitionIDs

Description: Returns the handles to all the field definitions defined on the given typeDefinitions element.

- Returns: fieldDefinitionIDs of type ***String*** - List of handles to the fieldDefinition elements
- Input: typeDefinitionsID of type ***String*** - Handle to a typeDefinitions element

F.7.81.36 getTypeDefinitionsMemoryMapDefinitionIDs

Description: Returns the handles to all the memoryMapDefinitions defined on the given typeDefinitions element.

- Returns: memoryMapDefinitionIDs of type ***String*** - List of handles to the memoryMapDefinition elements
- Input: typeDefinitionsID of type ***String*** - Handle to a typeDefinitions element

F.7.81.37 getTypeDefinitionsMemoryRemapDefinitionIDs

Description: Returns the handles to all the memoryRemapDefinitions defined on the given typeDefinitions element.

- Returns: memoryRemapDefinitionIDs of type ***String*** - List of handles to the memoryRemapDefinition elements
- Input: typeDefinitionsID of type ***String*** - Handle to a typeDefinitions element

F.7.81.38 getTypeDefinitionsModelIDs

Description: Returns the handles to all the modes defined on the given typeDefinitions element.

- Returns: modeIDs of type **String** - List of handles to the mode elements
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element

F.7.81.39 getTypeDefinitionsRegisterDefinitionIDs

Description: Returns the handles to all the registerDefinitions defined on the given typeDefinitions object.

- Returns: registerDefinitionIDs of type **String** - List of handles to the registerDefinition elements
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element

F.7.81.40 getTypeDefinitionsRegisterFileDefinitionIDs

Description: Returns the handles to all the RegisterFileDefinitions defined on the given typeDefinition element.

- Returns: registerFileDefinitionIDs of type **String** - List of handles to the registerFileDefinition elements
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element

F.7.81.41 getTypeDefinitionsResetTypeID

Description: Returns the handles to all the resetTypes defined on the given typeDefinitions element.

- Returns: resetTypesIDs of type **String** - List of handles to the resetTypes elements
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element

F.7.81.42 getTypeDefinitionsViewIDs

Description: Returns the handles to all the views defined on the given typeDefinitions element.

- Returns: viewIDs of type **String** - List of handles to the view elements
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element

F.7.81.43 getViewLinkExternalViewReferenceRefByName

Description: Returns the externalViewReference from the given viewLink element.

- Returns: externalViewReference of type **String** - The referenced externalView
- Input: viewLinkID of type **String** - Handle to a viewLink element

F.7.81.44 getViewLinkExternalViewReferenceID

Description: Returns the handle to the externalViewReference defined on the given viewLink element.

- Returns: externalViewReferenceID of type **String** - Handle to the externalViewReference element
- Input: viewLinkID of type **String** - Handle to a viewLink element

F.7.81.45 getViewLinkViewReferenceRefByID

Description: Returns the handle to the view referenced from the given viewLink element.

- Returns: viewID of type **String** - Handle to the referenced view element

- Input: viewLinkID of type **String** - Handle to a viewLink element

F.7.81.46 getViewLinkViewReferenceRefByName

Description: Returns the view referenced from the given viewLink element.

- Returns: viewRef of type **String** - The referenced view
- Input: viewLinkID of type **String** - Handle to a viewLink element

F.7.81.47 getViewLinkViewReferenceID

Description: Returns the handle to the viewReference defined on the given viewLink element.

- Returns: viewReferenceID of type **String** - Handle to the viewReference element
- Input: viewLinkID of type **String** - Handle to a viewLink element

F.7.82 Type definitions (EXTENDED)

F.7.82.1 addComponentExternalTypeDefinitions

Description: Adds a new ExternalTypeDefinitions to the given component.

- Returns: externalDefinitionsID of type **String** - The identifier of the added ExternalDefinitions
- Input: componentID of type **String** - Handle to the component type
- Input: name of type **String** - Name of the external type definition
- Input: vlnv of type **String[]** - Handle to the external type definition's VLNV

F.7.82.2 addEnumerationDefinition

Description: Adds an EnumerationDefinition to a ComponentType's EnumerationDefinitions and creates an underlying enumerated value.

- Returns: EnumerationDefinitionID of type **String** - Handle to the added EnumerationDefinition
- Input: typeDefID of type **String** - Handle to typeDef element
- Input: name of type **String** - Name of the new enumeration definition
- Input: enumeratedName of type **String** - Name of the (first) new enumeratedValue
- Input: enumeratedExpression of type **String** - Expression associated with the (first) new enumeratedValue

F.7.82.3 addEnumerationDefinitionEnumeratedValue

Description: Adds an EnumeratedValue to the given EnumerationDefinition element.

- Returns: EnumeratedValueID of type **String** - Handle to the added EnumeratedValue
- Input: enumerationDefinitionID of type **String** - Handle to an EnumerationDefinition
- Input: name of type **String** - Name of the new EnumeratedValue
- Input: value of type **String** - Expression of the new EnumeratedValue

F.7.82.4 addExternalTypeDefinitionsModeLink

Description: Adds the given modeLink in the given externalTypeDefinitions element.

- Returns: modeLinkID of type **String** - Handle to the added modeLink

- Input: `externalTypeDefinitionsID` of type **String** - Handle to an `externalTypeDefinitions` element
- Input: `externalModeRef` of type **String** - Name of the referenced external mode
- Input: `modeReference` of type **String** - the modeReferences to be add on the `modeLink`

F.7.82.5 addExternalTypeDefinitionsViewLink

Description: Adds the given `viewLink` in the given `externalTypeDefinitions` element.

- Returns: `viewLinkID` of type **String** - Handle to a `viewLink` element
- Input: `externalTypeDefinitionsID` of type **String** - Handle to an `externalTypeDefinitions` element
- Input: `externalViewRef` of type **String** - Name of the referenced external view
- Input: `viewReference` of type **String** - Name of the referenced view

F.7.82.6 addFieldDefinitionsEnumValue

Description: Adds an enumerated value (name, value pair) to the given `Field` definition.

- Returns: `enumeratedValueID` of type **String** - The identifier of the added `EnumeratedValue`
- Input: `fieldDefinitionID` of type **String** - Handle to the `fieldDefinition`
- Input: `name` of type **String** - `EnumeratedValue` name
- Input: `expression` of type **String** - `EnumeratedValue` value expression

F.7.82.7 addTypeDefinitionsAddressBlockDefinition

Description: Adds an `AddressBlockDefinition` to a `ComponentType`'s `AddressBlockDefinitions` and creates an underlying enumerated value.

- Returns: `addressBlockDefinitionID` of type **String** - Handle to the added `AddressBlockDefinition`
- Input: `typeDefinitionsID` of type **String** - Handle to `typeDefinitions` element
- Input: `name` of type **String** - Name of the new `AddressBlockDefinition`
- Input: `range` of type **String** - Range of the new `AddressBlockDefinition`
- Input: `width` of type **String** - Width of the new `AddressBlockDefinition`

F.7.82.8 addTypeDefinitionsBankDefinition

Description: Adds a `bankDefinition` to a `typeDefinitions` element.

- Returns: `bankDefinitionID` of type **String** - Handle to the added `bankDefinition`
- Input: `typeDefinitionsID` of type **String** - Handle to a `typeDefinitions` object
- Input: `name` of type **String** - Handle to a `bankDefinition` name

F.7.82.9 addTypeDefinitionsChoice

Description: Adds a choice with the given name and enumerations to the given `typeDefinitions` element.

- Returns: `choiceID` of type **String** - Handle to a new choice
- Input: `typeDefinitionsID` of type **String** - Handle to a `typeDefinitions` element
- Input: `name` of type **String** - Choice name
- Input: `enumerations` of type **String[]** - List of enumeration values

F.7.82.10 addTypeDefinitionsEnumerationDefinition

Description: Adds an enumerationDefinition to a typeDefinitions element.

- Returns: enumerationDefinitionID of type **String** - Handle to the added enumerationDefinition
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element
- Input: name of type **String** - EnumerationDefinition name
- Input: width of type **String** - width expression

F.7.82.11 addTypeDefinitionsExternalTypeDefinitions

Description: Adds an externalTypeDefinitions.

- Returns: externalTypeDefinitionsID of type **String** - Handle to the added externalTypeDefinitions
- Input: typeDefinitionsID of type **String** - Handle to the typeDefinitions
- Input: name of type **String** - Handle to an externalTypeDefinitions element
- Input: vlnv of type **String[]** - VLVN of the referenced typeDefinitions element

F.7.82.12 addTypeDefinitionsFieldAccessPolicyDefinition

Description: Adds a fieldAccessPolicyDefinition to a typeDefinitions element.

- Returns: fieldAccessPolicyDefinitionID of type **String** - Handle to the added fieldAccessPolicyDefinition
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element
- Input: name of type **String** - Handle to a fieldAccessPolicyDefinition name

F.7.82.13 addTypeDefinitionsFieldDefinition

Description: Adds a fieldDefinition to a component type.

- Returns: fieldDefinitionID of type **String** - Handle to the added fieldDefinition or null if the call failed
- Input: typeDefinitionsID of type **String** - Handle to the typeDefinitions type
- Input: name of type **String** - Handle to the name of the field definition
- Input: bitWidth of type **String** - Handle to the bitWidth expression associated with the field definition

F.7.82.14 addTypeDefinitionsMemoryMapDefinition

Description: Adds a memoryMapDefinition on the containing element.

- Returns: memoryMapDefinitionID of type **String** - The identifier of the added memoryMapDefinition
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element
- Input: name of type **String** - Name of the memoryMap

F.7.82.15 addTypeDefinitionsMemoryRemapDefinition

Description: Adds a memoryRemapDefinition to a typeDefinitions element.

- Returns: memoryRemapDefinitionID of type **String** - Handle to the added memoryRemapDefinition

- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element
- Input: name of type **String** - Handle to a memoryRemapDefinition name

F.7.82.16 addTypeDefinitionsMode

Description: Adds a mode to the given typeDefinitions element.

- Returns: modeID of type **String** - Handle to the added mode
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element
- Input: name of type **String** - Handle to Mode name

F.7.82.17 addTypeDefinitionsRegisterDefinition

Description: Adds a registerDefinition to a typeDefinition element.

- Returns: registerDefinitionID of type **String** - Handle of registerDefinition element
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element
- Input: name of type **String** - Handle to the registerDefintion name
- Input: size of type **String** - The value of the attribute Size
- Input: fieldName of type **String** - Field name
- Input: fieldOffset of type **String** - Field offset
- Input: fieldWidth of type **String** - Field width

F.7.82.18 addTypeDefinitionsRegisterFileDefinition

Description: Adds a registerFileDefinition to the given typeDefinitions element.

- Returns: registerFileDefinitionID of type **String** - Handle to the registerFileDefinition
- Input: typeDefinitionsID of type **String** - The identifier of a typeDefinitions
- Input: name of type **String** - The name of the RegisterFileDefinition to create
- Input: range of type **String** - The range expression associated with the registerFileDefinition to be created

F.7.82.19 addTypeDefinitionsResetType

Description: Adds a resetType on the given typeDefinition.

- Returns: resetTypeID of type **String** - ID of the created resetType
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions type
- Input: name of type **String** - Handle to the created resetType name

F.7.82.20 addTypeDefinitionsView

Description: Adds a view to the given typeDefinitions element.

- Returns: viewID of type **String** - Handle to the added view
- Input: typeDefinitionsID of type **String** - Handle to a typeDefinitions element
- Input: name of type **String** - Handle to Mode name

F.7.82.21 removeAddressBlockDefinitionAddressUnitBits

Description: Removes the addressUnitBits on an addressBlockDefinitionRef

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: addressBlockDefinitionID of type **String** - Handle to an addressBlockDefinition element

F.7.82.22 removeBankDefinitionAddressUnitBits

Description: Removes the addressUnitBits field of the given bankDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankDefinitionID of type **String** - Handle to a bankDefinition

F.7.82.23 removeComponentExternalTypeDefinitions

Description: Removes the given externalTypeDefinitions form the containing component element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: externalTypeDefinitionsID of type **String** - Handle to an externalTypeDefinitions element

F.7.82.24 removeEnumerationDefinitionEnumeratedValue

Description: Removes the given enumeratedValue from its containing enumerationDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: enumeratedValueID of type **String** - Handle to an enumeratedValue element

F.7.82.25 removeExternalTypeDefinitionsModeLink

Description: Removes the given modeLink from its containing externalTypeDefinitions element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeLinkID of type **String** - Handle to a modeLink element

F.7.82.26 removeExternalTypeDefinitionsResetTypeLink

Description: Removes the given resetLink from its containing externalTypeDefinitions element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: resetTypeLinkID of type **String** - Handle to a resetTypeLink element

F.7.82.27 removeExternalTypeDefinitionsViewLink

Description: Removes the given viewLink from its containing externalTypeDefinitions element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewLinkID of type **String** - Handle to a viewLink element

F.7.82.28 removeFieldDefinitionEnumerationDefinitionRef

Description: Removes the enumerationDefinitionRef from its containing fieldDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: fieldDefinitionID of type **String** - Handle to the enumerationDefinition

F.7.82.29 removeFieldDefinitionsEnumeratedValue

Description: Removes the given enumeratedValue from its containing fieldDefinitions element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: domainTypeDefID of type **String** - Handle to a domainTypeDef element

F.7.82.30 removeMemoryRemapDefinitionAddressUnitBits

Description: Removes the addressUnitBits field of the given memoryRemapDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryRemapDefinitionID of type **String** - Handle to a memoryRemapDefinition

F.7.82.31 removeRegisterFileDefinitionAddressUnitBits

Description: Removes the addressUnitBits field of the given registerFileDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileDefinitionID of type **String** - Handle to a registerFileDefinition

F.7.82.32 removeTypeDefinitionsAddressBlockDefinition

Description: Removes the given addressBlockDefinition from its containing typeDefinitions element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockDefinitionID of type **String** - Handle to an addressBlockDefinition object

F.7.82.33 removeTypeDefinitionsBankDefinition

Description: Removes the given bankDefinition from its containing typeDefinitions element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankDefinitionID of type **String** - Handle to a bankDefinition element

F.7.82.34 removeTypeDefinitionsChoice

Description: Removes the given choice element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: choiceID of type **String** - Handle to a choice element

F.7.82.35 removeTypeDefinitionsEnumerationDefinition

Description: Removes the given enumerationDefinition from its containing typeDefinitions element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: enumerationDefinitionID of type **String** - Handle to an enumerationDefinition element

F.7.82.36 removeTypeDefinitionsExternalTypeDefinitions

Description: Removes the given externalTypeDefinitions from its containing typeDefinitions element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: externalTypeDefinitionsID of type **String** - Handle to the externalTypeDefinitions

F.7.82.37 removeTypeDefinitionsFieldAccessPolicyDefinition

Description: Removes the given fieldAccessPolicyDefinition from its containing typeDefinitions element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)

- Input: `fieldAccessPolicyDefinitionID` of type ***String*** - Handle to a `fieldAccessPolicyDefinition` element

F.7.82.38 removeTypeDefinitionsFieldDefinition

Description: Removes the given `fieldDefinition` from its containing `typeDefinitions` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fieldDefinitionID` of type ***String*** - Handle to a `fieldDefinition` element

F.7.82.39 removeTypeDefinitionsMemoryMapDefinition

Description: Removes the given `memoryMapDefinition` from its containing `typeDefinitions` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `memoryMapDefinitionID` of type ***String*** - Handle to the aimed item

F.7.82.40 removeTypeDefinitionsMemoryRemapDefinition

Description: Removes the given `memoryRemapDefinition` from its containing `typeDefinitions` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `memoryRemapDefinitionID` of type ***String*** - Handle to a `memoryRemapDefinition` element

F.7.82.41 removeTypeDefinitionsMode

Description: Removes the given mode from its containing `typeDefinitions` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `modeID` of type ***String*** - Handle to a `mode` element

F.7.82.42 removeTypeDefinitionsRegisterDefinition

Description: Removes the given `registerDefinition` from its containing `typeDefinitions` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerDefinitionID` of type ***String*** - Handle to a `registerDefinition`

F.7.82.43 removeTypeDefinitionsRegisterFileDefinition

Description: Removes the given `registerFileDefinition` from its containing `typeDefinitions` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileDefinitionID` of type ***String*** - Handle to a `registerFileDefinition` element

F.7.82.44 removeTypeDefinitionsResetType

Description: Removes the given `resetType` from its containing `typeDefinitions` element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `resetTypeID` of type ***String*** - Handle to the `resetType` element

F.7.82.45 removeTypeDefinitionsView

Description: Removes the given `view` from its containing `typeDefinitions` object.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewID of type **String** - Handle to a view element

F.7.82.46 setAddressBlockDefinitionAddressUnitBits

Description: Sets the addressUnitBits on an addressBlockDefinitionRef.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: addressBlockDefinitionID of type **String** - Handle to an addressBlockDefinition element
- Input: addressUnitBits of type **String** - The addressUnitBits expression

F.7.82.47 setBankDefinitionAddressUnitBits

Description: Sets the addressUnitBits field of the given bankDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: bankDefinitionID of type **String** - Handle to a bankDefinition
- Input: addressUnitBits of type **String** - Handle to the new value for the addressUnitBits field (expression / string)

F.7.82.48 setEnumerationDefinitionWidth

Description: Sets the width field of the given enumerationDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: enumerationDefinitionID of type **String** - Handle to a typeDefinitions element
- Input: width of type **String** - Handle to the new value for the width field (expression/string)

F.7.82.49 setExternalTypeDefinitionsTypeDefinitionsRef

Description: Sets the typeDefinitionsRef on the externalTypeDefinitions.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: externalTypeDefinitionsID of type **String** - Handle to an externalTypeDefinitions element
- Input: typeDefinitionsRef of type **String[]** - VLVN of the referenced typeDefinitions object

F.7.82.50 setMemoryRemapDefinitionAddressUnitBits

Description: Sets the addressUnitBits field of the given memoryRemapDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: memoryRemapDefinitionID of type **String** - Handle to a memoryRemapDefinition
- Input: addressUnitBits of type **String** - Handle to the new value for the addressUnitBits field (expression / string)

F.7.82.51 setModeLinkExternalModeReference

Description: Sets the externalModeReference on a modeLink element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeLinkID of type **String** - Handle to a modeLink element
- Input: modeRef of type **String** - the modeRef value to set on the externalModeReference

F.7.82.52 setModeLinkModeReference

Description: Sets the modeReference on the given modeLink element

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: modeLinkID of type **String** - Handle to a modeLink element
- Input: modeRef of type **String** - Handle to a modeRef element

F.7.82.53 setRegisterFileDefinitionAddressUnitBits

Description: Sets the addressUnitBits field of the given registerFileDefinition element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: registerFileDefinitionID of type **String** - Handle to a registerFileDefinition
- Input: addressUnitBits of type **String** - Handle to the new value for the addressUnitBits field (expression / string)

F.7.82.54 setResetTypeLinkExternalResetTypeReference

Description: Sets the externalResetTypeReference of a resetTypeLink element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: resetTypeLinkID of type **String** - Handle to a resetTypeLink element
- Input: resetTypeRef of type **String** - the value of the resetTypeRef on the externalResetTypeReference

F.7.82.55 setResetTypeLinkResetTypeReference

Description: Sets the resetTypeReference on a resetTypeLink element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: resetTypeLinkID of type **String** - Handle to a resetTypeLink element
- Input: resetTypeRef of type **String** - the value of the resetTypeRef on the externalResetTypeReference

F.7.82.56 setViewLinkExternalViewReference

Description: Sets the externalViewReference on a viewLink elements

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewLinkID of type **String** - Handle to a viewLink element
- Input: viewRef of type **String** - The viewRef expression to be set on the externalViewReference

F.7.82.57 setViewLinkViewReference

Description: Sets the viewReference on a viewLink elements

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewLinkID of type **String** - Handle to a viewLink element
- Input: viewRef of type **String** - The viewRef expression to be set on the viewReference

F.7.83 Vector (BASE)

F.7.83.1 getVectorIDs

Description: Returns the handles to all the vectors defined on the given element.

- Returns: `vectorIDs` of type ***String*** - List of handles to vector elements
- Input: `vectorContainerID` of type ***String*** - Handle to an element that has a vector element

F.7.83.2 getVectorLeftID

Description: Returns the handle to the left range of the given vector element.

- Returns: `LeftID` of type ***String*** - Handle to the left element
- Input: `vectorID` of type ***String*** - Handle to a vector element

F.7.83.3 getVectorRange

Description: Returns the range defined on the given vector element.

- Returns: `range` of type ***Long*** - Array of two range values: left and right
- Input: `vectorID` of type ***String*** - Handle to a vector element

F.7.83.4 getVectorRangeExpression

Description: Returns the range expressions defined on the given vector element.

- Returns: `rangeExpression` of type ***String*** - Array of two range expressions: left and right
- Input: `vectorID` of type ***String*** - Handle to a vector element

F.7.83.5 getVectorRightID

Description: Returns the handle to the right range of the given vector element.

- Returns: `RightID` of type ***String*** - Handle to the right element
- Input: `vectorID` of type ***String*** - Handle to a vector element

F.7.84 Vector (EXTENDED)

F.7.84.1 addVector

Description: Adds a vector to the given element.

- Returns: `vectorID` of type ***String*** - Handle to new vector
- Input: `vectorContainerElementID` of type ***String*** - Handle to an element that has a vector element
- Input: `range` of type ***String[]*** - Range expression with left in index 0 and right in index 1

F.7.84.2 removeVector

Description: Removes the given vector.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `vectorID` of type ***String*** - Handle to a vector element

F.7.85 Vendor extensions (BASE)

F.7.85.1 getVendorExtensions

Description: Returns the handle to the vendorExtension defined on the given element.

- Returns: vendorExtensions of type **String** - Handle to the vendorExtension
- Input: vendorExtensionsContainerElementID of type **String** - Handle to an element that has a vendor extension element

F.7.86 Vendor extensions (EXTENDED)

F.7.86.1 addVendorExtensions

Description: Adds a vendor extension with given value to the given element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: vendorExtensionsContainerElementID of type **String** - Handle to an element that has a vendor extension element
- Input: vendorExtensions of type **String** - Vendor extension value

F.7.86.2 removeVendorExtensions

Description: Removes the given vendor extension.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: vendorExtensionsContainerElementID of type **String** - Handle to an element that has a vendor extension element

F.7.86.3 setVendorExtensions

Description: Sets the vendorExtension XML string to given container element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: vendorExtensionsContainerElementID of type **String** - Handle to an element that has a vendor extension element
- Input: vendorExtensions of type **String** - Vendor extension value

F.7.87 View (BASE)

F.7.87.1 getAbstractorInstanceViewNameRefByID

Description: Returns the handle to the view referenced from the given abstractorInstance element.

- Returns: viewID of type **String** - Handle to the referenced view element
- Input: abstractorInstanceID of type **String** - Handle to an abstractorInstance element

F.7.87.2 getViewComponentInstantiationRefByID

Description: Returns the handle to the componentInstantiation referenced from the given view element.

- Returns: componentInstantiationID of type **String** - Handle to the referenced componentInstantiation element
- Input: viewID of type **String** - Handle to the view element

F.7.87.3 getViewComponentInstantiationRefByName

Description: Returns the componentInstantiationRef from the given view element.

- Returns: componentInstantiationRef of type **String** - The referenced componentInstantiation name
- Input: viewID of type **String** - Handle to a view element

F.7.87.4 getViewDesignConfigurationInstantiationID

Description: Returns the handle to a designConfigurationInstantiation from the given view element.

- Returns: designConfigurationInstantiationID of type **String** - Handle to a designConfigurationInstantiation
- Input: viewID of type **String** - Handle to a view element

F.7.87.5 getViewDesignConfigurationInstantiationRefByID

Description: Returns the handle to the designConfigurationInstantiation referenced from the given view element.

- Returns: designConfigurationInstantiationID of type **String** - Handle to the referenced designConfigurationInstantiation element
- Input: viewID of type **String** - Handle to a view element

F.7.87.6 getViewDesignConfigurationInstantiationRefByName

Description: Returns the designConfigurationInstantiationRef from the given view element.

- Returns: designConfigurationInstantiationRef of type **String** - The referenced designConfigurationInstantiation name
- Input: viewID of type **String** - Handle to a view element

F.7.87.7 getViewDesignInstantiationRefByID

Description: Returns the handle to the designInstantiation referenced from the given view element.

- Returns: designInstantiationID of type **String** - Handle to the referenced designInstantiation element
- Input: viewID of type **String** - Handle to the view element

F.7.87.8 getViewDesignInstantiationRefByName

Description: Returns the designInstantiation reference from the given view element.

- Returns: designInstantiationRef of type **String** - The referenced designInstantiation name
- Input: viewID of type **String** - Handle to a view element

F.7.87.9 getViewEnvIdentifierIDs

Description: Returns the handles to all the envIdentifiers from the given view element.

- Returns: envIdentifierIDs of type **String** - List of handles to the envIdentifier elements
- Input: viewID of type **String** - Handle to a view

F.7.87.10 getViewEnvIdentifiers

Description: Returns the envIdentifiers defined on the given view element.

- Returns: `envIdentifier` of type ***String*** - List of view envIdentifiers
- Input: `viewID` of type ***String*** - Handle to a view element

F.7.87.11 getViewRefByName

Description: Returns the view name defined on the given viewRef element.

- Returns: `viewName` of type ***String*** - The referenced view
- Input: `viewRefID` of type ***String*** - Handle to a viewRef element

F.7.88 View (EXTENDED)

F.7.88.1 addViewEnvIdentifier

Description: Adds an envIdentifier on the given view of a component or on a view of an abstractor.

- Returns: `envIdentifierID` of type ***String*** - Handle to the added envIdentifier
- Input: `viewID` of type ***String*** - Handle to a view element
- Input: `value` of type ***String*** - Name of the envIdentifier

F.7.88.2 removeViewComponentInstantiationRef

Description: Removes componentInstantiationRef for the given view element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `viewID` of type ***String*** - Handle to a view element

F.7.88.3 removeViewDesignConfigurationInstantiationRef

Description: Removes designConfigurationInstantiationRef for the given view element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `viewID` of type ***String*** - Handle to a view element

F.7.88.4 removeViewDesignInstantiationRef

Description: Removes designInstantiationRef for the given view element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `viewID` of type ***String*** - Handle to a view element

F.7.88.5 removeViewEnvIdentifier

Description: Removes the given envIdentifier from its containing a view element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `envIdentifierID` of type ***String*** - Handle to an envIdentifier element

F.7.88.6 setViewComponentInstantiationRef

Description: Sets componentInstantiationRef for the given view element.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

- Input: viewID of type **String** - Handle to a view element
- Input: componentInstantiationRef of type **String** - ComponentInstantiation name

F.7.88.7 setViewDesignConfigurationInstantiationRef

Description: Sets designConfigurationInstantiationRef for the given view element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewID of type **String** - Handle to a view element
- Input: designInstantiationRef of type **String** - designConfigurationInstantiation name

F.7.88.8 setViewDesignInstantiationRef

Description: Sets designInstantiationRef for the given view element.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewID of type **String** - Handle to a view element
- Input: designInstantiationRef of type **String** - designInstantiation name

F.7.88.9 setViewRefValue

Description: Sets the name of the viewRef.

- Returns: status of type **Boolean** - Indicates call is successful (true) or not (false)
- Input: viewRefID of type **String** - Handle to a viewRef element
- Input: value of type **String** - Name of the referenced view

F.7.89 All ID types

This subclause lists the TGI ID types as follows:

- | | |
|------------------------------------|-------------------------------|
| — absDefOrAbsDefInstanceID | — addressBlockID |
| — absDefPortID | — addressBlockRefID |
| — abstractionDefID | — addressSpaceID |
| — abstractionDefPortID | — addressSpaceRefID |
| — abstractionDefPortModeID | — aliasOfID |
| — abstractionDefinitionID | — alternateRegisterID |
| — abstractionTypeID | — alternateRegisterRefID |
| — abstractorBusInterfaceID | — argumentID |
| — abstractorGeneratorID | — arraryID |
| — abstractorID | — arrayContainerElementID |
| — abstractorInstanceID | — arrayID |
| — abstractorInstancesID | — arrayOrfieldArrayID |
| — abstractorInterfaceID | — assertionContainerElementID |
| — abstractorOrAbstractorInstanceID | — assertionID |
| — abstractorViewIDaccessHandleID | — attributeContainerID |
| — accessID | — bankDefinitionID |
| — accessPolicyID | — bankID |
| — accessRestrictionID | — bankOrLocalBankID |
| — activeInterfaceID | — bankRefID |
| — adHocConnectionID | — bankrefID |
| — adHocExternalPortReferenceID | — baseAddressesID |
| — adHocInternalPortReferenceID | — broadcastToID |
| — addressBlockDefinitionID | — buildCommandID |
| — addressBlockDefinitionRefID | — busDefID |

— busDefOrBusDefInstanceID	— fieldAccessPoliciesID
— busDefinitionID	— fieldDefinitionID
— busInterfaceID	— fieldID
— busInterfaceRefID	— fieldMapID
— catalogID	— fieldRefID
— cellSpecificationID	— fieldSliceID
— chainGroupID	— fileBuilderID
— channelID	— fileDefineID
— choiceEnumerationID	— fileID
— choiceID	— fileSetID
— clearboxElementID	— fileSetRefGroupID
— clearboxElementRefID	— fileSetRefID
— clearboxElementRefLocationID	— fileTypeID
— clearboxID clockDriverID	— functionID
— clockPeriodID	— functionSourceFileID
— componentGeneratorID	— generatorChainConfigurationID
— componentGeneratorSelectorID	— generatorChainID
— componentID	— generatorChainSelectorID
— componentInstanceID	— generatorID
— componentInstantiationID	— generatorRefID
— componentOrComponentInstanceID	— groupID
— configurableElementID	— groupSelectorID
— configurableElementValueID	— groupSelectorNameID
— configuredElementID	— hierInterfaceID
— constraintSetID	— imageTypeID
— constraintSetRefID	— indexID
— constraintSetID	— indirectAddressRefID
— cpuID	— indirectDataRefID
— defaultFileBuilderID	— indirectInterfaceID
— defineID	— initiatorID
— dependencyID	— interconnectionConfigurationID
— designConfID	— interconnectionID
— designConfigurationID	— interfaceRefID
— designConfigurationInstantiationID	— internalPortReferenceID
— designConfigurationOrDesignConfigurationInstan-	— ipxactFileID
— ceID	— languageToolsID
— designID	— linkerCommandFileID
— designInstantiationID	— loadConstraintID
— dimID	— localMemoryMapID
— domainTypeDefID	— locationID
— driveConstraintID	— logicalPortID
— driverID	— memoryMapDefinitionID
— elementContainerID	— memoryMapElementID
— elementID	— memoryMapID
— enumeratedValueID	— memoryMapOrLocalMemoryMapID
— enumeratedValuesID	— memoryMapRefID
— enumerationDefinitionID	— memoryRemapDefinitionID
— envIdentifierID	— memoryRemapID
— excludePortID	— memoryMapID
— executableImageID	— mirroredSystemID
— exportedNameID	— mirroredTargetID
— expressionID	— modeConstraintsID
— externalPortReferenceID	— modeConstraintsID
— externalTypeDefinitionsID	— modeID
— fieldAccessPoliciesID	— modeLinkID
— fieldAccessPolicyDefinitionID	— modeRefID
— fieldAccessPolicyID	— moduleOrTypeParameterContainerElementID

— moduleParameterID	— resetTypeLinkID
— moduleParameterTypeID	— rootObjectID
— monitorID	— segmentID
— monitorInterconnectionID	— serviceTypeDefID
— monitorInterfaceID	— signalTypeDefID
— monitoredActiveInterfaceID	— singleShotDriverID
— onSystemID	— sliceID
— otherClockDriverID	— sourceFileID
— packetFieldID	— strucPortTypeDefID
— packetID	— structPortTypeDefID
— parameterBaseTypeID	— structuredID
— parameterContainerElementID	— subPortID
— parameterID	— subPortMapID
— parametrizedID	— subPortRefID
— partSelectID	— subPortReferenceID
— pathSegmentID	— subSpaceID
— payLoadID	— subSpaceMapID
— payloadID	— subspaceMapID
— physicalPortID	— systemGroupNameID
— portID	— systemID
— portMapID	— targetID
— portModeID	— timingConstraintID
— portSliceID	— transTypeDefID
— portWireID	— transactionalID
— powerConstraintID	— transparentBridgeID
— powerDomainID	— transparentBrigeID
— powerDomainLinkID	— transportMethodsID
— powerDomainRefID	— typeDefID
— powerEnID	— typeDefinitionID
— protocolID	— typeDefinitionsID
— qualifierID	— typeParameterID
— referenceID	— unconfiguredElementID
— regFieldID	— vectorContainerElementID
— regionID	— vectorContainerID
— registerDefinitionID	— vectorID
— registerFieldID	— vendorExtensionsContainerElementID
— registerFileDefinitionID	— viewConfigurationID
— registerFileID	— viewConfigurationOrViewID
— registerFileRefID	— viewID
— registerID	— viewLinkID
— registerRefID	— viewRefID
— remapAddressID	— wireID
— remapAddressesID	— wireTypeDefID
— resetID	— writeValueConstraintID
— resetTypeID	

Annex G

(informative)

External bus with an internal/digital interface

While the current use of IP-XACT schema may be viewed as describing single chip implementations, the schemas works equally well at the package- and board-level. Often a **PHY** component exists that interconnects the internal and external bus. Some interface standards define both of these interfaces, some define only the internal, and some define only the external. A common point of confusion is to use an external bus standard as an interface on an internal component. This is legal if the component carries the full **PHY** implementation, but this often makes the component very technology- or implementation-dependent.

G.1 Example: Ethernet interfaces

An Ethernet bus might be described as more than a single wire, and in a system that includes Ethernet buses, it might also include all the interfaces shown in [Figure G.1](#).

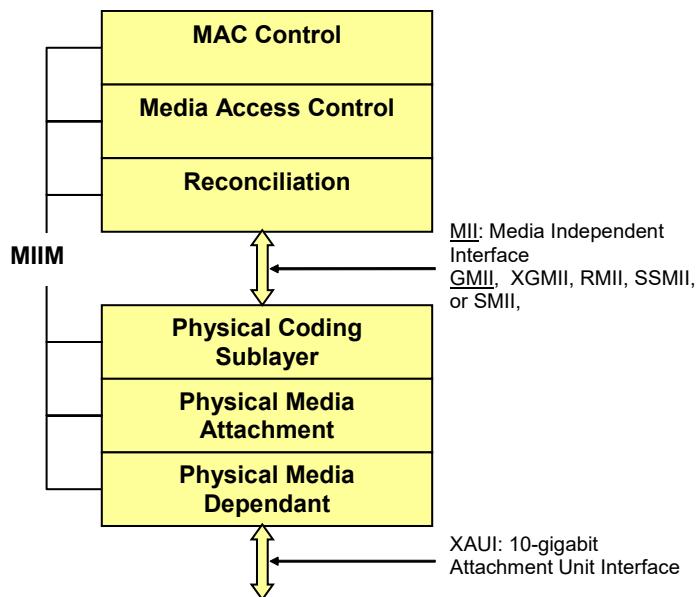


Figure G.1—Ethernet interface examples

XAUI: 10-gigabit Attachment Unit Interface

MII: Media-Independent Interface

GMII: Gigabit Media-Independent Interface

XGMII: 10-gigabit media-independent interface

RMII: Reduced MII, 7-pin interface

SSMII: Source Synchronous MII

SMII: Serial Media-Independent Interface, this provides an interface to Ethernet MAC. The SMII provides the same interface as the MII, but with a reduced pin-out. The reduction in ports is achieved by multiplexing data and control information to a port transmit port and a single receive port.

G.2 Example: I²C bus

The I²C bus is a two-wire bus with a clock and data line. The standard described bus is the two-wire bus. IP-XACT has defined an additional, related bus that is the internal digital interface. The internal digital interface shown in [Figure G.2](#) contains three pins for each external pin: for SDA (the data line), the internal pins are defined as input, output, and enable as SDA_I, SDA_O, and SDA_E; in a similar manner, for the clock bus SCL, the internal pins are defined again for the functions of input, output, and enable as SCL_I, SCL_O, and SCL_E.

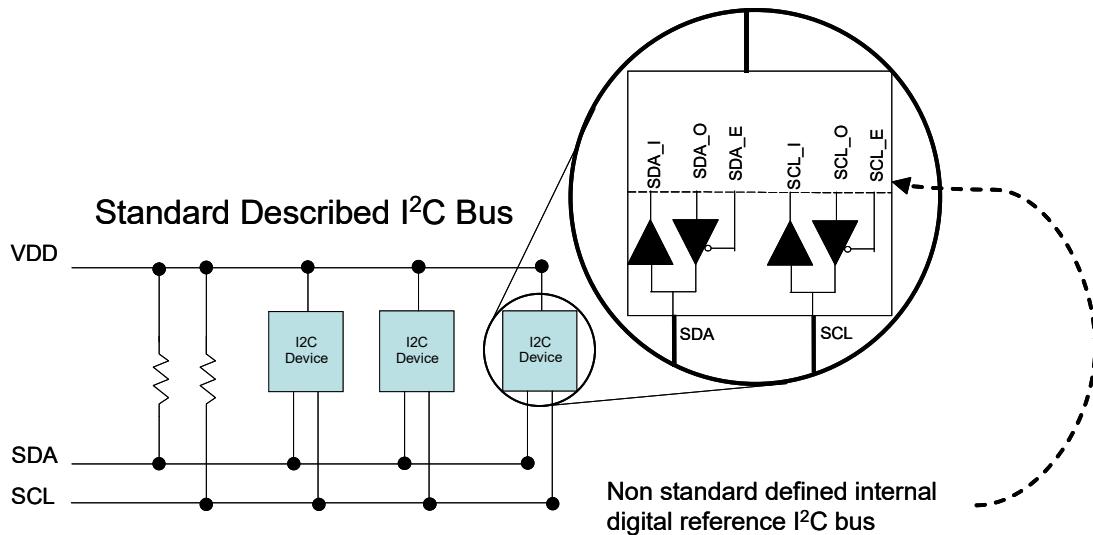


Figure G.2—I²C interface example

Annex H

(informative)

Bridges and channels

This annex describes the basic address calculations of the two interconnect schemes contained inside an IP-XACT component: a **bridge** statement that describes an interconnect between a target interface and a initiator interface, and a **channel** statement that describes an interconnect between a mirrored-initiator interface and a mirrored-target interface. [Figure H.1](#) highlights bridge and channel components in IP-XACT. For precise details on the addressing equations, see [Clause 13](#).

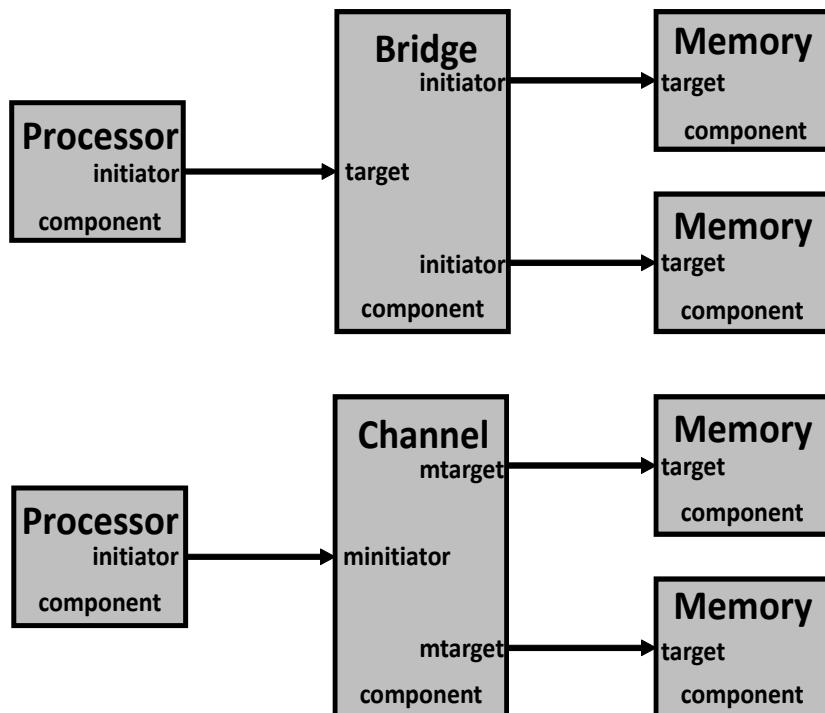


Figure H.1—Bridge and channel components

H.1 Transparent bridge

A transparent bridge locates the start of the initiator interface's address space at the start of the address space seen at the target interface; thus, the address is not modified from the target interface of the bridge into the **addressSpace** of the initiator interface. In [Figure H.2](#), the initiator interface address space range `'h0000` to `'h0FFF` maps to the address range `'h0000` to `'h0FFF` as seen in the address space at the target interface.

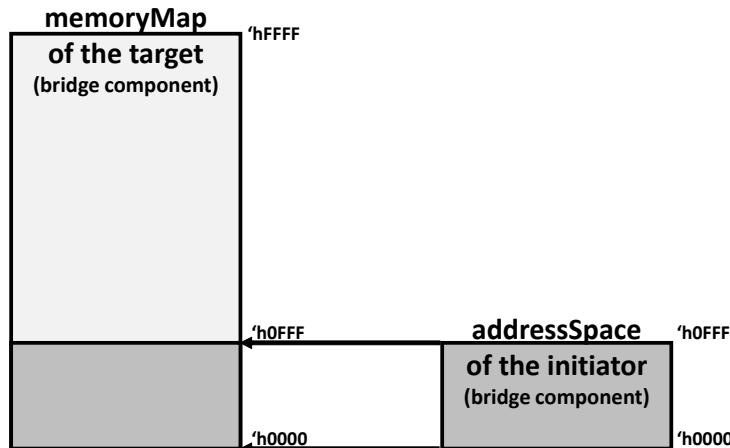


Figure H.2—Transparent bridge target interface address range

An address block from another component connected to the bridge's initiator interface may appear in the initiator's address space. The base address of the connected address block is offset in the address space of the target interface by the **initiator/addressSpaceRef/baseAddress**.

In [Figure H.3](#), the **addressBlock** from the connected target range `'h0000` to `'h07FF` maps to the address range `'h0000` to `'h07FF` as seen in the address space of the initiator interface and to the address range `'h0600` to `'h0DFF` (offset by **initiator/addressSpaceRef/baseAddress** = `'h0600`) as seen in the address space at the target interface.

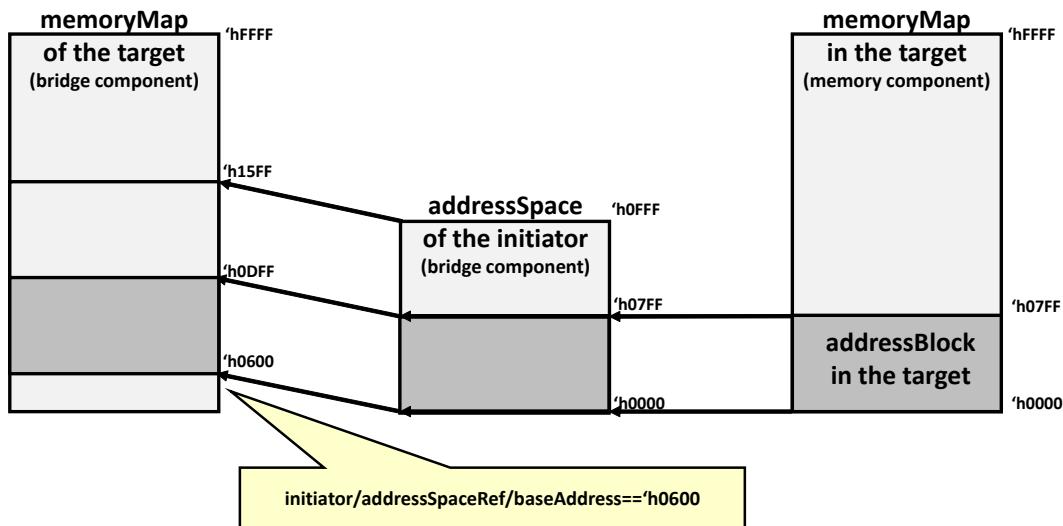


Figure H.3—Offsetting an address block in a transparent bridge

[Figure H.4](#) shows it is also possible to offset the **addressBlock** from the connected target in the negative direction. The **addressBlock** from the connected target range `'h7000` to `'h77FF` maps to the address range `'h7000` to `'h77FF` as seen in the address space of the initiator interface and to the address range `'h0000` to `'h07FF` (offset by **initiator/addressSpaceRef/baseAddress** = `-'h7000`) as seen in the address space at the target interface.interface.

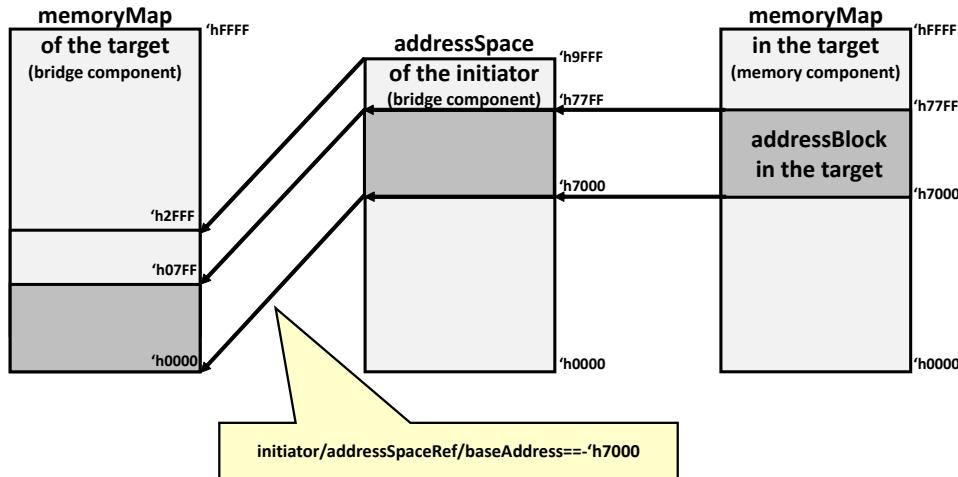


Figure H.4—Negative offsetting of an address block in a transparent bridge

[Figure H.5](#) shows the references between the various elements and attributes in a transparent bridge.

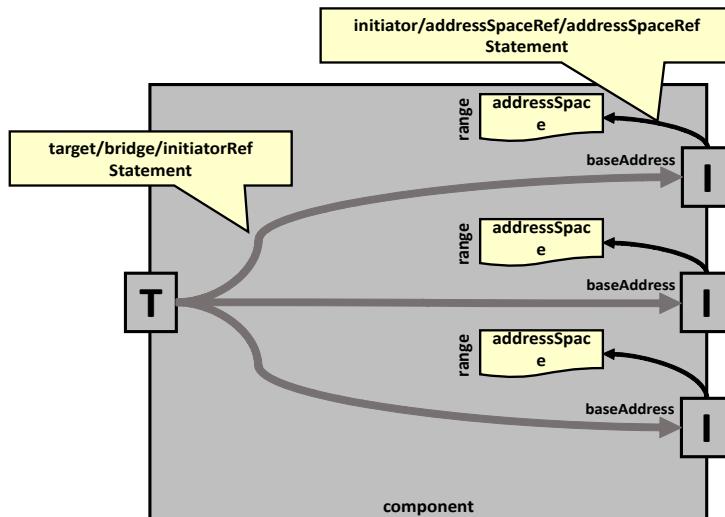


Figure H.5—Transparent bridge references

H.2 Opaque bridge

H.2.1 Without an address space segment reference

An opaque bridge that references only an initiator interface locates the start of the initiator interface's address space at the base address specified in the subspace map referenced by the target interface; thus, the

address is modified (offset by **subspaceMap/baseAddress**) from the target interface into the **addressSpace** of the initiator interface. In [Figure H.6](#), the target interface address range `'h1000` to `'h1FFF` maps to address range `'h0000` to `'h0FFF` in the initiator interface's address space. The **range** of the addresses mapped is determined by the **range** of the **addressSpace**.

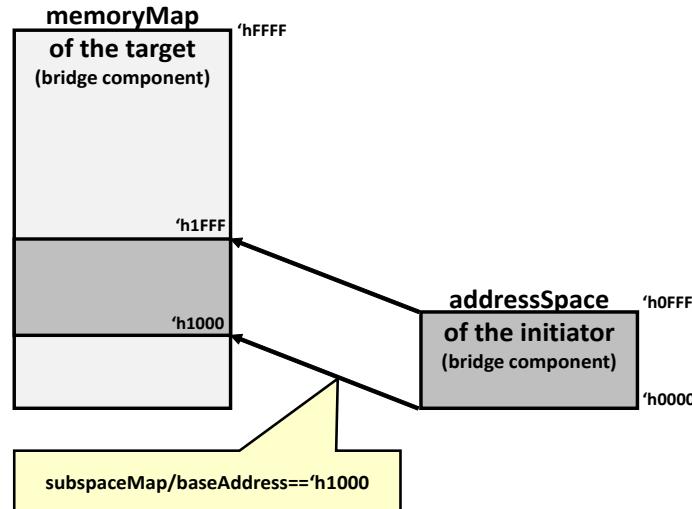


Figure H.6—Opaque bridge target interface address range without segment reference

H.2.2 With an address space segment reference

An opaque bridge with an **addressSpace** segment reference locates the start of the initiator interface's address space segment at the base address specified in the subspace map referenced by the target interface; thus, the address is modified (offset by **subspaceMap/baseAddress**) from the target interface into the **addressSpace** of the initiator interface. In [Figure H.7](#), the target interface address range `'h1000` to `'h17FF` maps to address range `'h2000` to `'h27FF` in the initiator interface's address space. The **range** of the addresses mapped is determined by the **range** of the address space's segment.

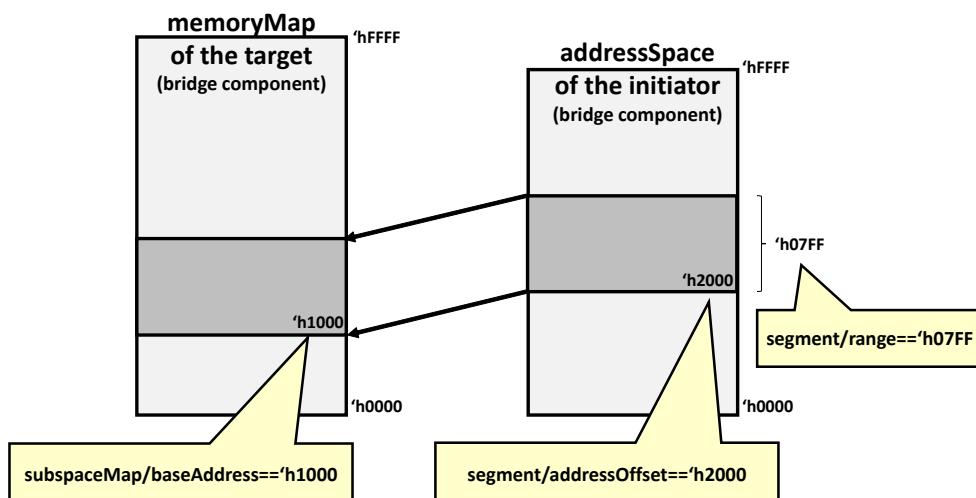


Figure H.7—Opaque bridge target interface address range with segment reference

It is also possible to preserve the addressing across an opaque bridge. In [Figure H.8](#), the target interface address range 'h1000 to 'h17FF maps to address range 'h1000 to 'h17FF in the initiator interface's address space. The **range** of the addresses mapped is determined by the **range** of the address space's space's segment.

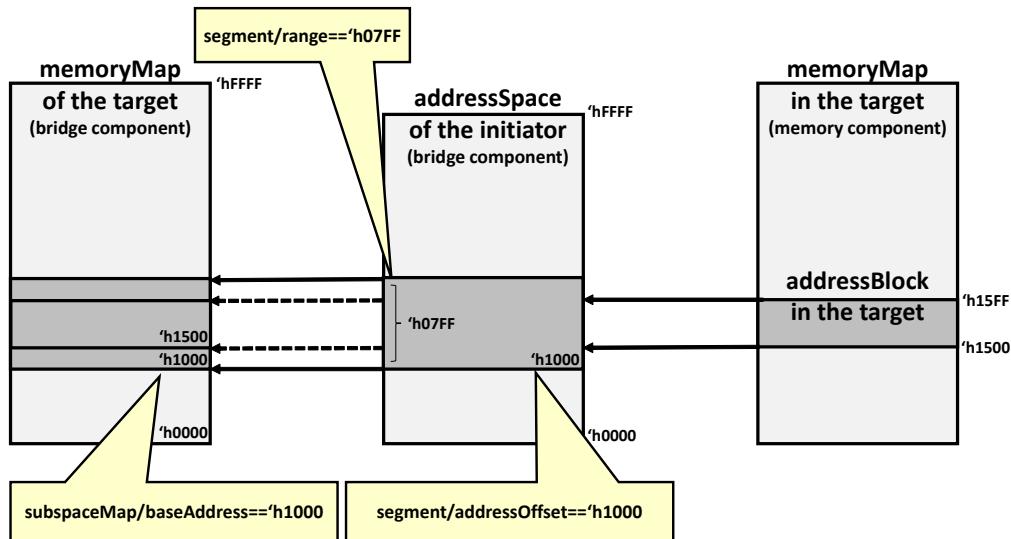


Figure H.8—Opaque bridge with transparent addressing

H.2.3 Effect of an initiator interface address space base address

The effect of the initiator interface address space base address applies with or without a segment reference. An address block from another component connected to the bridge's initiator interface may appear in the initiator's address space. The base address of the connected address block is offset in the address space of the initiator interface. Here the **initiator/addressSpaceRef/baseAddress** is ignored. This also offsets the address block by the same amount in the address space at the target interface. In [Figure H.9](#), the **addressBlock** from the connected target range '`h0000` to '`h07FF` maps to the address range '`h0000` to '`h07FF` (offset by **initiator/addressSpaceRef/baseAddress** = '`h0500` is ignored) as seen in the address

space of the initiator interface and to the address range 'h1500 to 'h1CFF (offset by **subspaceMap/baseAddress** = 'h1000) as seen in the address space at the target interface.

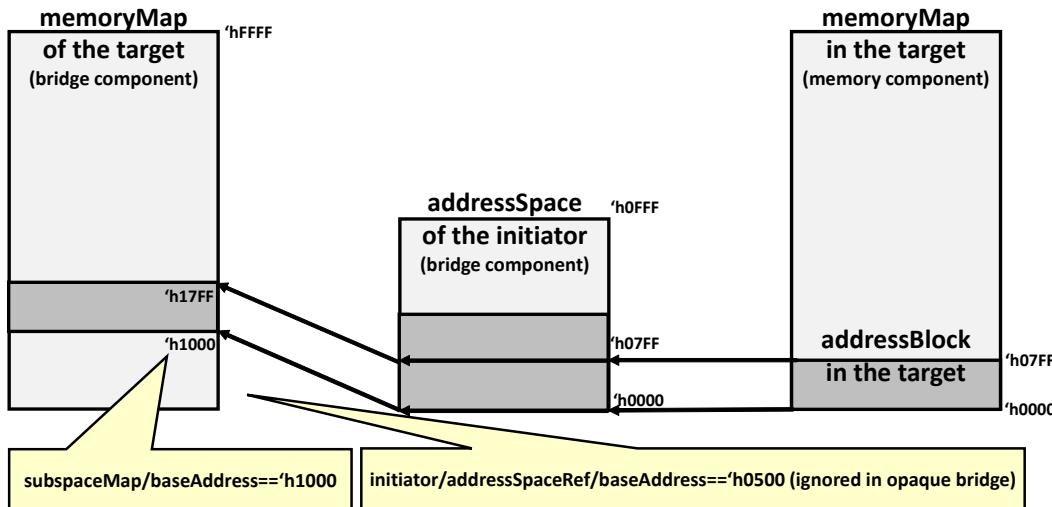


Figure H.9—Opaque bridge with transparent addressing

[Figure H.10](#) shows it is also possible to offset the **addressBlock** from the connected target in the negative direction. The **addressBlock** from the connected target range 'h7000 to 'h77FF maps to the address range 'h7000 to 'h77FF (offset by **initiator/addressSpaceRef/baseAddress** = -0x6B00 is ignored) as seen in the address space of the initiator interface and to the address range 0x1500 to 0x1CFF (offset by **subspaceMap/baseAddress** = 0x1000) as seen in the address space at the target interface.

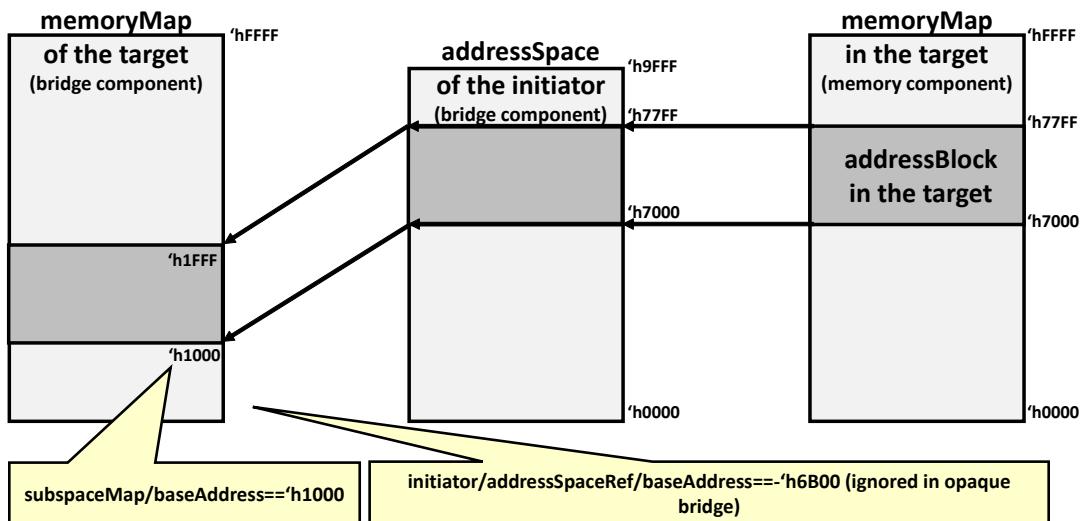


Figure H.10—Opaque bridge with transparent addressing

[Figure H.11](#) provides a reference for an opaque bridge.

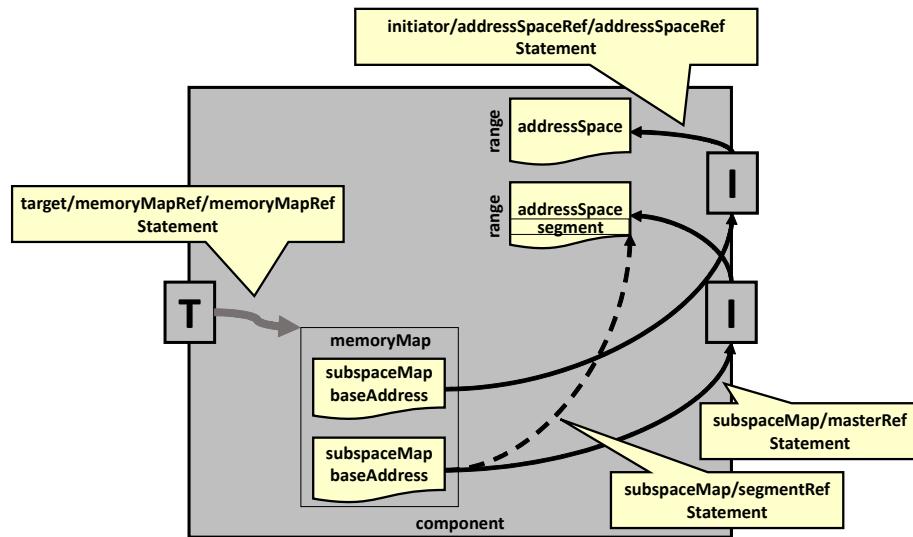


Figure H.11—Opaque bridge references

H.3 Channel with address remapping

A mirrored-target interface that is part of a channel may provide a remap address for the connected target interface. This remap address is an offset of the base address of the address block in the connected target interface, as shown in [Figure H.12](#). This offset is the addition the **remapAddress** element's value (see [6.13](#)) to the base address of the memory map from the target. The **range** element also modifies (and potentially narrows) the range of the entire memory map of the connected target. In [Figure H.12](#), the target interface address range 'h0000' to 'h0FFF' maps to the address range 'h1000' to 'h17FF' (offset by **mirrorTarget/baseAddress/remapAddress** = 'h1000' and narrowed by **mirrorTarget/baseAddress/range** = 'h0800) in the address space as seen at the mirrored-target interface.

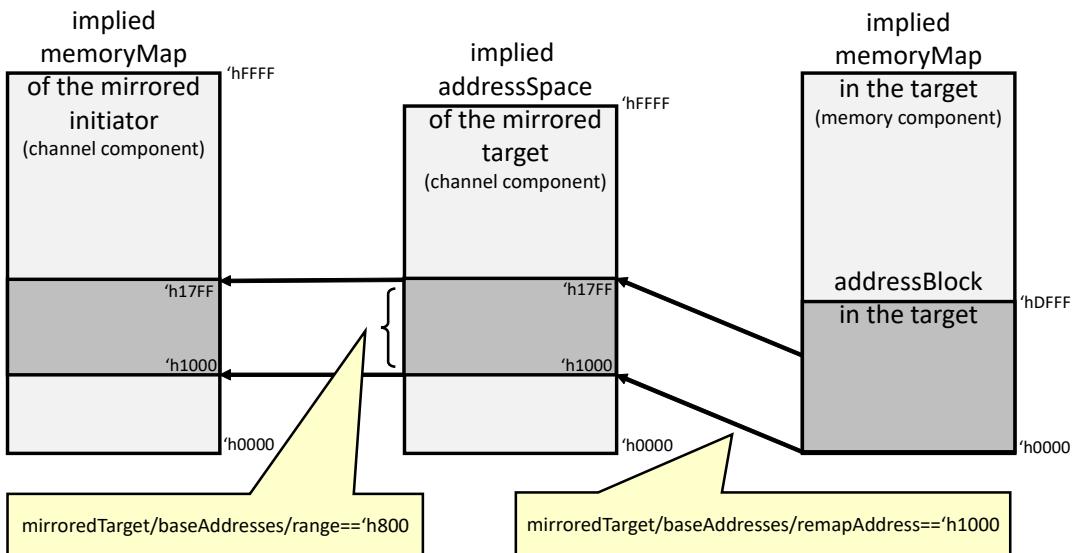


Figure H.12—Address remapping in a channel

[Figure H.13](#) shows the references between the various elements and attributes in a channel with address remapping.

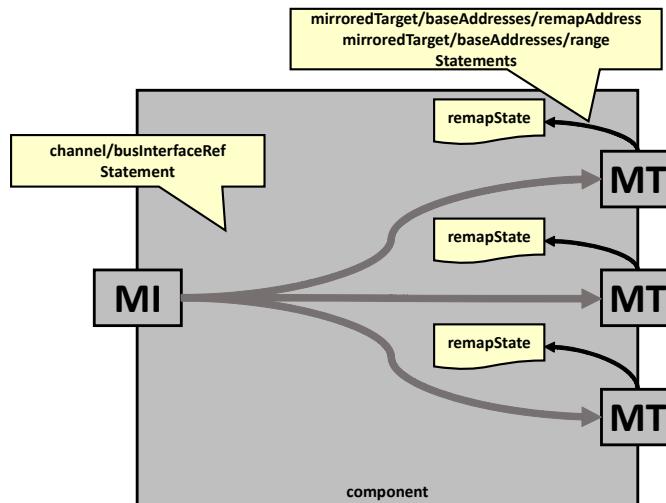


Figure H.13—Channel with remapping references

H.4 Channel with bit steering

A mirrored-target interface that is part of a channel may have a **bitSteering** element. If the **bitSteering** element is 1, the base address and range of an address block as seen across a channel are modified only by the ratio of the **bitsInLau** of the mirrored-target and the mirrored-initiator interfaces. In [Figure H.14](#), the target interface address range 'h0000 to 'h07FF maps to the address range 'h1000 to 'h17FF (offset by **mirrorTarget/baseAddress/remapAddress** = 'h1000) in the address space as seen at the mirrored-target interface and maps to the address range 'h1000 to 'h17FF (multiplied by the ratio of the mirrored-target and mirrored-initiator **bitsInLau** 8/8 = 1, and independent of the width of the logical data ports) in the address space as seen at the mirrored-initiator interface.

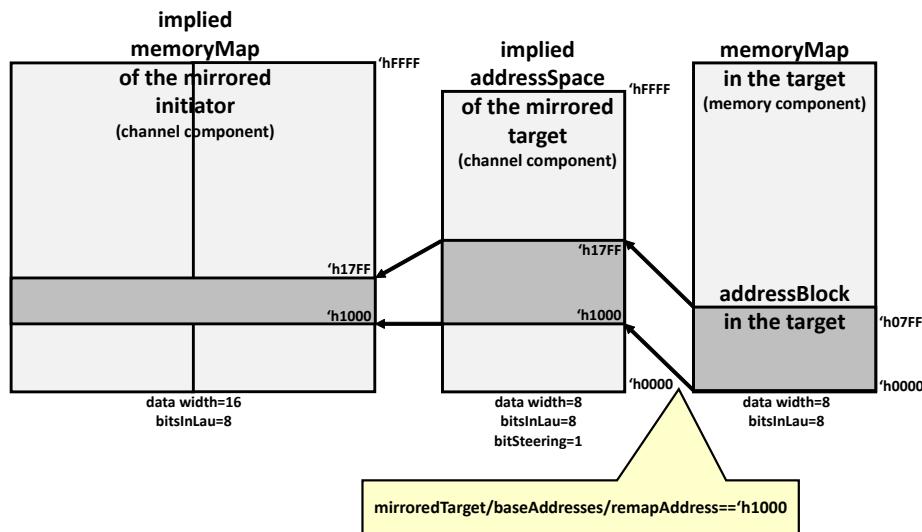


Figure H.14—Channel with equal **bitsInLau** and **bitSteering** = 1

In [Figure H.15](#), the **bitSteering** element is 1, and there are differing **bitsInLau** across the channel. The target interface address range 'h0000 to 'h0FFF maps to the address range 'h1000 to 'h1FFF (offset by **mirrorTarget/baseAddress/remapAddress** = 'h1000) in the address space as seen at the mirrored-target interface and maps to the address range 'h0800 to 'h0FFF (multiplied by the ratio of the mirrored-target and mirrored-initiator **bitsInLau** 4/8 = 1/2, and independent of the width of the logical data ports) in the address space as seen at the mirrored-initiator interface.

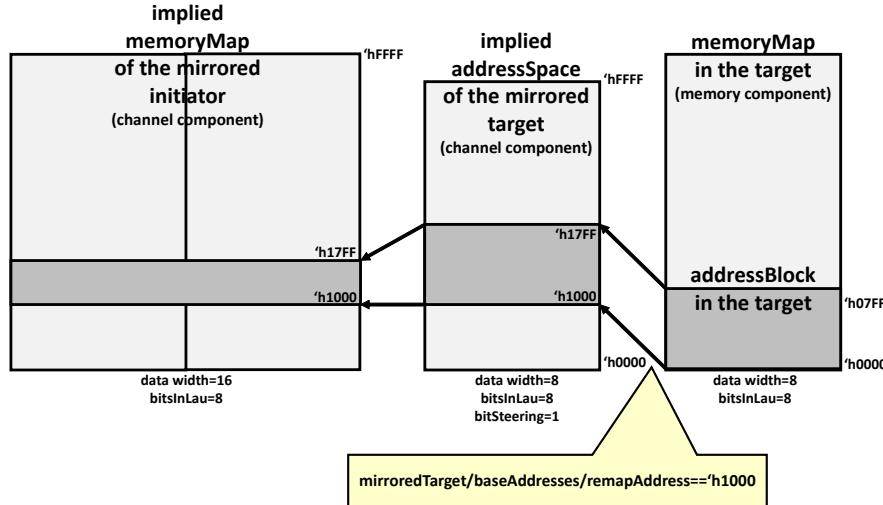


Figure H.15—Channel with non-equal bitsInLau and bitSteering = 1

If **bitSteering** is 0, the base address and range of an address block as seen across a channel is modified by the ratio of the **bitsInLau** and the logical data-width of the mirrored-target and mirrored-initiator interfaces. In [Figure H.16](#), the target interface address range 'h0000 to 'h07FF maps to the address range 'h1000 to 'h17FF (offset by **mirrorTarget/baseAddress/remapAddress** = 'h1000) in the address space as seen at the mirrored-target interface and maps to the address range 'h2000 to 'h2FFF (multiplied by the ratio of mirrored-target and mirrored-initiator **bitsInLau** 8/8 = 1, and multiplied by the ratio mirrored-initiator logical data-width divided by the mirrored-target logical data-width 16/8 = 2) in the address space as seen at the mirrored-initiator interface.

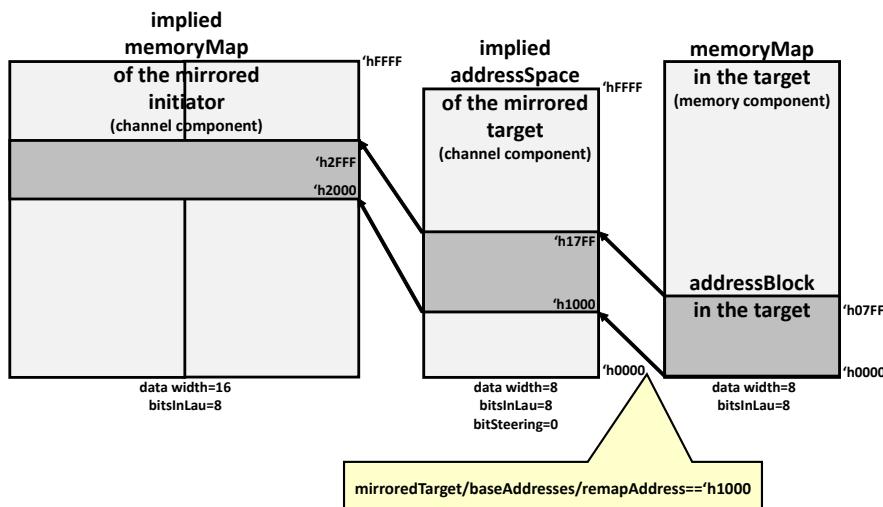


Figure H.16—Channel with bitSteering = 0

[Figure H.17](#) shows the references between the various elements and attributes in a channel with **bitSteering** references.

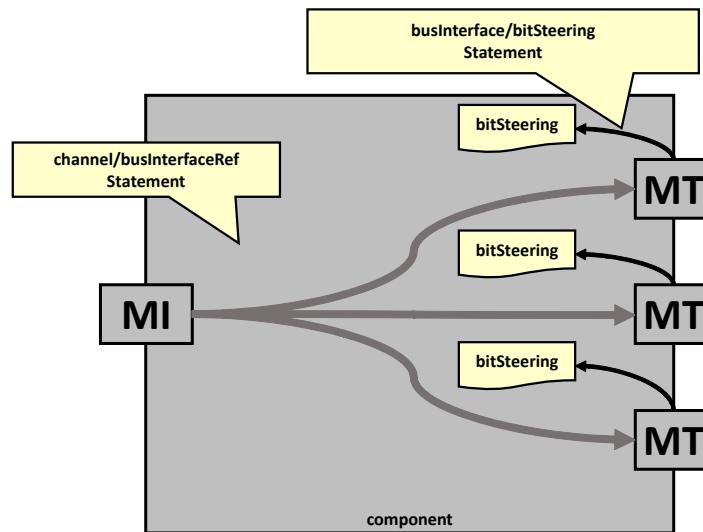


Figure H.17—Channel with bitSteering references

Annex I

(informative)

Examples

This annex shows a set of examples covering some practical uses of IP-XACT (syntax) and a series of pointers to the various syntax definitions, as appropriate. Within each example, any `LINK:` comments show the relevant [cross-reference](#) to that element's image/descriptions.

I.1 abstractionDefinition - RTL

```
<?xml version="1.0"?>
<!--
// Example abstraction definition used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document. This file represents an RTL abstraction.
-->
<!-- LINK: abstractionDefinition: see 5.3, Abstraction definition -->
<ipxact:abstractionDefinition xmlns:ipxact="http://www.accellera.org/
    XMLSchema/IPXACT/1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/
    1685-2022 http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
    <!-- LINK: documentNameGroup: C.11, documentName group -->
    <ipxact:vendor>accellera.org</ipxact:vendor>
    <ipxact:library>Sample</ipxact:library>
    <ipxact:name>SampleAbstractionDefinition_RTL</ipxact:name>
    <ipxact:version>1.0</ipxact:version>
    <ipxact:description>Example RTL abstraction definition used in the IP-XACT
    standard.</ipxact:description>
    <ipxact:busType vendor="accellera.org" library="Sample"
    name="SampleBusDefinitionExtension" version="1.0"/>
    <ipxact:ports>
        <!-- LINK: port: see 5.4, Ports -->
        <!-- Simple wire port -->
        <ipxact:port>
            <ipxact:logicalName>Data</ipxact:logicalName>
            <ipxact:displayName>Data Bus</ipxact:displayName>
            <ipxact:shortDescription>The data bus</ipxact:shortDescription>
            <ipxact:description>This port carries the data</ipxact:description>
            <ipxact:match>false</ipxact:match>
            <!-- LINK: wire: see 5.5, Wire ports -->
            <ipxact:wire>
                <!-- LINK: qualifier: see 5.6, Qualifiers -->
                <ipxact:qualifier>
                    <ipxact:isData>true</ipxact:isData>
                </ipxact:qualifier>
                <ipxact:onInitiator>
                    <!-- LINK: wirePort: see 5.7, Wire port group -->
                    <ipxact:presence>required</ipxact:presence>
                    <ipxact:width>dataWidth</ipxact:width>
                    <ipxact:direction>out</ipxact:direction>
                    <!-- LINK: modeConstraints: see 5.8, Wire port mode (and
                    mirrored mode) constraints -->
                    <ipxact:modeConstraints>
```

```

        <ipxact:timingConstraint clockName="Clk">40</
    ipxact:timingConstraint>
        </ipxact:modeConstraints>
    </ipxact:onInitiator>
    <ipxact:onTarget>
        <ipxact:width>dataWidth</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
    </ipxact:onTarget>
        <ipxact:defaultValue>'ff</ipxact:defaultValue>
    </ipxact:wire>
    <!-- Link: packets: see 5.11, Packets -->
    <ipxact:packets>
        <ipxact:packet>
            <ipxact:name>frame</ipxact:name>
            <ipxact:endianness>little</ipxact:endianness>
            <ipxact:packetFields>
                <ipxact:packetField>
                    <ipxact:name>data_bits</ipxact:name>
                    <ipxact:width>dataWidth-1</ipxact:width>
                    <ipxact:qualifier>
                        <ipxact:isData>true</ipxact:isData>
                    </ipxact:qualifier>
                </ipxact:packetField>
                <ipxact:packetField>
                    <ipxact:name>parity_bit</ipxact:name>
                    <ipxact:width>1</ipxact:width>
                </ipxact:packetField>
            </ipxact:packetFields>
        </ipxact:packet>
    </ipxact:packets>
</ipxact:port>
<ipxact:port>
    <ipxact:logicalName>Address</ipxact:logicalName>
    <ipxact:displayName>Address Bus</ipxact:displayName>
    <ipxact:shortDescription>The address bus</ipxact:shortDescription>
    <ipxact:description>This port carries the address</
    ipxact:description>
    <!-- LINK: wire: see 5.5, Wire ports -->
    <ipxact:wire>
        <!-- LINK: qualifier: see 5.6, Qualifiers -->
        <ipxact:qualifier>
            <ipxact:isAddress>true</ipxact:isAddress>
        </ipxact:qualifier>
        <ipxact:onInitiator>
            <!-- LINK: wirePort: see 5.7, Wire port group -->
            <ipxact:presence>required</ipxact:presence>
            <ipxact:direction>out</ipxact:direction>
        </ipxact:onInitiator>
        <ipxact:onTarget>
            <ipxact:width>8</ipxact:width>
            <ipxact:direction>in</ipxact:direction>
        </ipxact:onTarget>
            <ipxact:defaultValue>'ff</ipxact:defaultValue>
        </ipxact:wire>
    </ipxact:port>
    <!-- Simple clock port to show system elements -->
    <ipxact:port>
        <ipxact:logicalName>Clk</ipxact:logicalName>
        <ipxact:wire>

```

```

<ipxact:qualifier>
    <ipxact:isClock>true</ipxact:isClock>
</ipxact:qualifier>
<ipxact:onSystem>
    <ipxact:group>SystemSignals</ipxact:group>
    <ipxact:width>1</ipxact:width>
    <ipxact:direction>out</ipxact:direction>
</ipxact:onSystem>
<ipxact:onInitiator>
    <ipxact:presence>optional</ipxact:presence>
    <ipxact:width>1</ipxact:width>
    <ipxact:direction>in</ipxact:direction>
</ipxact:onInitiator>
<ipxact:onTarget>
    <ipxact:presence>optional</ipxact:presence>
    <ipxact:width>1</ipxact:width>
    <ipxact:direction>in</ipxact:direction>
</ipxact:onTarget>
    <ipxact:requiresDriver driverType="singleShot">true</
ipxact:requiresDriver>
    <ipxact:wire>
        </ipxact:port>
    </ipxact:ports>
<ipxact:parameters>
    <ipxact:parameter parameterId="dataWidth" type="int"
resolve="generated">
        <ipxact:name>dataWidth</ipxact:name>
        <ipxact:description>Number of data bits.</ipxact:description>
        <ipxact:value>32</ipxact:value>
    </ipxact:parameter>
</ipxact:parameters>
</ipxact:abstractionDefinition>

```

I.2 abstractionDefinition - TLM

```

<?xml version="1.0"?>
<!--
// Example abstraction definition used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document. This is a trivial TLM abstraction.
-->
<!-- LINK: abstractionDefinition: see 5.3, Abstraction definition -->
<ipxact:abstractionDefinition xmlns:ipxact="http://www.accellera.org/
XMLSchema/IPXACT/1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/
1685-2022 http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
<!-- LINK: documentNameGroup: C.11, documentName group -->
<ipxact:vendor>accellera.org</ipxact:vendor>
<ipxact:library>Sample</ipxact:library>
<ipxact:name>SampleAbstractionDefinition_TLM</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<ipxact:description>Example TLM abstraction definition used in the IP-XACT
standard.</ipxact:description>
<ipxact:busType vendor="accellera.org" library="Sample"
name="SampleBusDefinitionExtension" version="1.0"/>
<ipxact:ports>
    <!-- LINK: port: see 5.4, Ports -->

```

```

<!-- Simple transactional port -->
<ipxact:port>
    <ipxact:logicalName>Transport</ipxact:logicalName>
    <!-- LINK: transactional: see 5.9, Transactional ports -->
    <ipxact:transactional>
        <ipxact:onInitiator>
            <!-- abstractionDefinition/transactionalPort -->
            <ipxact:presence>optional</ipxact:presence>
            <ipxact:initiative>provides</ipxact:initiative>
            <ipxact:kind>simple_socket</ipxact:kind>
            <ipxact:busWidth>8</ipxact:busWidth>
            <!-- LINK: protocol: see 6.15.20, Component transactional
            protocol/payload definition -->
            <ipxact:protocol>
                <ipxact:protocolType>tlm</ipxact:protocolType>
                <ipxact:payload>
                    <ipxact:name>tlm2</ipxact:name>
                    <ipxact:type>generic</ipxact:type>
                </ipxact:payload>
            </ipxact:protocol>
        </ipxact:onInitiator>
        <ipxact:onTarget>
            <ipxact:initiative>requires</ipxact:initiative>
            <ipxact:kind>simple_socket</ipxact:kind>
            <ipxact:protocol>
                <ipxact:protocolType>tlm</ipxact:protocolType>
                <ipxact:payload>
                    <ipxact:name>tlm2</ipxact:name>
                    <ipxact:type>generic</ipxact:type>
                </ipxact:payload>
            </ipxact:protocol>
        </ipxact:onTarget>
    </ipxact:transactional>
</ipxact:port>
</ipxact:ports>
</ipxact:abstractionDefinition>

```

I.3 abstractor

```

<?xml version="1.0"?>
<!--
// Example abstractor used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document.
-->
<!-- LINK: abstractor: see 8.1, Abstractor -->
<ipxact:abstractor xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/
1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
<!-- LINK: documentNameGroup: C.11, documentName group -->
<ipxact:vendor>accellera.org</ipxact:vendor>
<ipxact:library>Sample</ipxact:library>
<ipxact:name>SampleAbstractor</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<ipxact:description/>
<ipxact:abstractorMode>initiator</ipxact:abstractorMode>

```

```

<ipxact:busType vendor="accelera.org" library="Sample"
    name="SampleBusDefinition" version="1.0"/>
<!-- LINK: abstractorInterfaces: see 8.2, Abstractor interfaces -->
<ipxact:abstractorInterfaces>
    <!-- initiator interface -->
    <ipxact:abstractorInterface>
        <ipxact:name>Initiator</ipxact:name>
        <ipxact:abstractionTypes>
            <ipxact:abstractionType>
                <ipxact:abstractionRef vendor="accelera.org"
                    library="Sample" name="SampleAbstractionDefinition_TLM" version="1.0"/>
                </ipxact:abstractionType>
            </ipxact:abstractionTypes>
        </ipxact:abstractorInterface>
        <!-- mirrored initiator interface -->
        <ipxact:abstractorInterface>
            <ipxact:name>MirroredInitiator</ipxact:name>
            <ipxact:abstractionTypes>
                <ipxact:abstractionType>
                    <ipxact:abstractionRef vendor="accelera.org"
                        library="Sample" name="SampleAbstractionDefinition_RTL" version="1.0"/>
                    </ipxact:abstractionType>
                </ipxact:abstractionTypes>
            </ipxact:abstractorInterface>
        </ipxact:abstractorInterface>
    <!-- Remaining content in abstractor is optional and similar enough -->
    <!-- to a component that it is excluded from the example -->
</ipxact:abstractor>

```

I.4 busDefinition

```

<?xml version="1.0"?>
<!--
// Example bus definition used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document
-->
<!-- LINK: abstractionDefinition: see 5.3, Abstraction definition -->
<ipxact:busDefinition xmlns:ipxact="http://www.accelera.org/XMLSchema/
    IPXACT/1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.accelera.org/XMLSchema/IPXACT/1685-2022
    http://www.accelera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
    <!-- LINK: documentNameGroup: C.11, documentName group -->
    <ipxact:vendor>accelera.org</ipxact:vendor>
    <ipxact:library>Sample</ipxact:library>
    <ipxact:name>SampleBusDefinitionExtension</ipxact:name>
    <ipxact:version>1.0</ipxact:version>
    <ipxact:description>Example bus definition used in the IP-XACT standard.</
    ipxact:description>
    <ipxact:directConnection>true</ipxact:directConnection>
    <ipxact:broadcast>true</ipxact:broadcast>
    <ipxact:isAddressable>false</ipxact:isAddressable>
    <ipxact:extends vendor="accelera.org" library="Sample"
        name="SampleBusDefinitionBase" version="1.0"/>
    <ipxact:maxInitiators>1</ipxact:maxInitiators>
    <ipxact:maxTargets>16</ipxact:maxTargets>
    <ipxact:systemGroupNames>

```

```

<ipxact:systemGroupName>SystemSignals</ipxact:systemGroupName>
</ipxact:systemGroupNames>
<!-- ipxact:parameters - excluded for brevity - see component example -->
<!-- ipxact:assertions - excluded for brevity - see component example -->
<!-- ipxact:vendorExtensions - excluded for brevity - see component example
    -->
</ipxact:busDefinition>

```

I.5 catalog

```

<?xml version="1.0"?>
<!--
// Example catalog used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document
-->
<!-- LINK: catalog: see 12.1, catalog -->
<ipxact:catalog xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-
2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
<!-- LINK: documentNameGroup: C.10, documentName group -->
<ipxact:vendor>accellera.org</ipxact:vendor>
<ipxact:library>Sample</ipxact:library>
<ipxact:name>SampleCatalog</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<ipxact:description>Example catalog used in the IP-XACT standard.</
ipxact:description>
<ipxact:busDefinitions>
    <!-- LINK: catalog/ipxactFile: see 12.2, ipxactFile -->
    <ipxact:ipxactFile>
        <ipxact:vlnv vendor="accellera.org" library="Sample"
name="SampleBusDefinitionBase" version="1.0"/>
            <ipxact:name>./SampleBusDefinitionBase.xml</ipxact:name>
            <ipxact:description>File included for semantic validation only.</
ipxact:description>
        </ipxact:ipxactFile>
        <ipxact:ipxactFile>
            <ipxact:vlnv vendor="accellera.org" library="Sample"
name="SampleBusDefinitionExtension" version="1.0"/>
                <ipxact:name>./SampleBusDefinitionExtension.xml</ipxact:name>
                <ipxact:description>Sample extended busDefinition</
ipxact:description>
            </ipxact:ipxactFile>
        </ipxact:ipxactFile>
    </ipxact:busDefinitions>
    <ipxact:abstractionDefinitions>
        <ipxact:ipxactFile>
            <ipxact:vlnv vendor="accellera.org" library="Sample"
name="SampleAbstractionDefinition_RTL" version="1.0"/>
                <ipxact:name>./SampleAbstractionDefinition_RTL.xml</ipxact:name>
            </ipxact:ipxactFile>
            <ipxact:ipxactFile>
                <ipxact:vlnv vendor="accellera.org" library="Sample"
name="SampleAbstractionDefinition_TLM" version="1.0"/>
                    <ipxact:name>./SampleAbstractionDefinition_TLM.xml</ipxact:name>
            </ipxact:ipxactFile>
        </ipxact:ipxactFile>
    </ipxact:abstractionDefinitions>

```

```
<!-- ipxact:components -->
<ipxact:components>
    <ipxact:ipxactFile>
        <ipxact:vlnv vendor="accelera.org" library="Sample"
name="SampleComponent" version="1.0"/>
        <ipxact:name>./SampleComponent.xml</ipxact:name>
    </ipxact:ipxactFile>
</ipxact:components>
<!-- ipxact:abstractors -->
<ipxact:abstractors>
    <ipxact:ipxactFile>
        <ipxact:vlnv vendor="accelera.org" library="Sample"
name="SampleAbstractor" version="1.0"/>
        <ipxact:name>./SampleAbstractor.xml</ipxact:name>
    </ipxact:ipxactFile>
</ipxact:abstractors>
<!-- ipxact:designs -->
<ipxact:designs>
    <ipxact:ipxactFile>
        <ipxact:vlnv vendor="accelera.org" library="Sample"
name="SampleDesign" version="1.0"/>
        <ipxact:name>./SampleDesign.xml</ipxact:name>
    </ipxact:ipxactFile>
</ipxact:designs>
<!-- ipxact:designConfigurations -->
<ipxact:designConfigurations>
    <ipxact:ipxactFile>
        <ipxact:vlnv vendor="accelera.org" library="Sample"
name="SampleDesignConfiguration" version="1.0"/>
        <ipxact:name>./SampleDesignConfiguration.xml</ipxact:name>
    </ipxact:ipxactFile>
</ipxact:designConfigurations>
<!-- ipxact:generatorChains -->
<ipxact:generatorChains>
    <ipxact:ipxactFile>
        <ipxact:vlnv vendor="accelera.org" library="Sample"
name="SampleGeneratorChain" version="1.0"/>
        <ipxact:name>./SampleGeneratorChain.xml</ipxact:name>
    </ipxact:ipxactFile>
    <ipxact:ipxactFile>
        <ipxact:vlnv vendor="accelera.org" library="Sample"
name="SampleGeneratorChainForReference" version="1.0"/>
        <ipxact:name>./SampleGeneratorChainForReference.xml</ipxact:name>
    </ipxact:ipxactFile>
</ipxact:generatorChains>
<!-- ipxact:typeDefinitions -->
<ipxact:typeDefinitions>
    <ipxact:ipxactFile>
        <ipxact:vlnv vendor="accelera.org" library="Sample"
name="SampleTypeDefinitions" version="1.0"/>
        <ipxact:name>./SampleTypeDefinitions.xml</ipxact:name>
    </ipxact:ipxactFile>
</ipxact:typeDefinitions>
</ipxact:catalog>
```

I.6 component

```
<?xml version="1.0"?>
<!--
// Example component used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document.
-->
<!-- LINK: component: see 6.1, Component -->
<ipxact:component xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/
1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
<!-- LINK: documentNameGroup: C.10, documentName group -->
<ipxact:vendor>accellera.org</ipxact:vendor>
<ipxact:library>Sample</ipxact:library>
<ipxact:name>SampleComponent</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<ipxact:displayName>Sample Component</ipxact:displayName>
<ipxact:shortDescription>Example component</ipxact:shortDescription>
<ipxact:description>Example component used in the IP-XACT standard.</
ipxact:description>
<!-- LINK: typeDefinitions: 6.2, Type definitions -->
<ipxact:typeDefinitions>
    <ipxact:externalTypeDefinitions>
        <ipxact:name>sampleTypes</ipxact:name>
        <ipxact:typeDefinitionsRef vendor="accellera.org" library="Sample"
name="SampleTypeDefinitions" version="1.0">
            <ipxact:configurableElementValues>
                <ipxact:configurableElementValue referenceId="addrBits">16</
ipxact:configurableElementValue>
            </ipxact:configurableElementValues>
        </ipxact:typeDefinitionsRef>
        <ipxact:viewLinks>
            <ipxact:viewLink>
                <!-- Reference view in external typeDefinitions file -->
                <ipxact:externalViewReference viewRef="RTLview"/>
                <!-- Reference view in this component -->
                <ipxact:viewReference viewRef="RTLview"/>
            </ipxact:viewLink>
        </ipxact:viewLinks>
        <ipxact:modeLinks>
            <ipxact:modeLink>
                <!-- Reference mode in external typeDefinitions file -->
                <ipxact:externalModeReference modeRef="Normal"/>
                <!-- Reference mode in this component -->
                <ipxact:modeReference modeRef="Normal"/>
            </ipxact:modeLink>
        </ipxact:modeLinks>
        <ipxact:resetTypeLinks>
            <ipxact:resetTypeLink>
                <!-- Reference resetType in external typeDefinitions file -->
                <ipxact:externalResetTypeReference resetTypeRef="SOFT"/>
                <!-- Reference resetType in this component -->
                <ipxact:resetTypeReference resetTypeRef="SOFT"/>
            </ipxact:resetTypeLink>
        </ipxact:resetTypeLinks>
    </ipxact:externalTypeDefinitions>
```

```

</ipxact:typeDefinitions>
<!-- LINK: powerDomains: 6.3, Power domains -->
<ipxact:powerDomains>
    <ipxact:powerDomain>
        <ipxact:name>PD1</ipxact:name>
        <ipxact:description>Power domain target side</ipxact:description>
    </ipxact:powerDomain>
    <ipxact:powerDomain>
        <ipxact:name>PD2</ipxact:name>
        <ipxact:description>Power domain initiator side</
ipxact:description>
    </ipxact:powerDomain>
</ipxact:powerDomains>
<!-- LINK: busInterfaces: 6.7, Bus interfaces -->
<ipxact:busInterfaces>
    <!-- LINK: busInterface: 6.7.1, busInterface -->
    <!-- Basic initiator interface with both RTL and TLM representations -->
    <ipxact:busInterface>
        <ipxact:name>Initiator</ipxact:name>
        <ipxact:busType vendor="accellera.org" library="Sample">
name="SampleBusDefinition" version="1.0"/>
        <!-- LINK: abstractionTypes: 6.7.2, Abstraction types -->
        <ipxact:abstractionTypes>
            <ipxact:abstractionType>
                <ipxact:viewRef>RTLview</ipxact:viewRef>
                <ipxact:abstractionRef vendor="accellera.org" library="Sample" name="SampleAbstractionDefinition_RTL" version="1.0">
                    <!-- Configure abstraction to have data width 8 -->
                    <ipxact:configurableElementValues>
                        <ipxact:configurableElementValue
referenceId="dataWidth">8</ipxact:configurableElementValue>
                    </ipxact:configurableElementValues>
                </ipxact:abstractionRef>
                <ipxact:portMaps>
                    <!-- LINK: portMap: 6.7.3, Port map -->
                    <ipxact:portMap>
                        <ipxact:logicalPort>
                            <ipxact:name>Data</ipxact:name>
                        </ipxact:logicalPort>
                        <ipxact:physicalPort>
                            <ipxact:name>init_data</ipxact:name>
                        </ipxact:physicalPort>
                    </ipxact:portMap>
                    <ipxact:portMap>
                        <ipxact:logicalPort>
                            <ipxact:name>Address</ipxact:name>
                        </ipxact:logicalPort>
                        <ipxact:physicalPort>
                            <ipxact:name>init_addr</ipxact:name>
                        </ipxact:physicalPort>
                    </ipxact:portMap>
                    <!-- Mapping to clock port included for 'information' only
- not for connecting -->
                    <ipxact:portMap>
                        <ipxact:logicalPort>
                            <ipxact:name>Clk</ipxact:name>
                        </ipxact:logicalPort>
                        <ipxact:physicalPort>
                            <ipxact:name>clk</ipxact:name>
                        </ipxact:physicalPort>
                    </ipxact:portMap>
                </ipxact:portMaps>
            </ipxact:abstractionType>
        </ipxact:abstractionTypes>
    </ipxact:busInterface>
</ipxact:busInterfaces>

```

```

        </ipxact:physicalPort>
        <ipxact:isInformative>true</ipxact:isInformative>
    </ipxact:portMap>
</ipxact:portMaps>
</ipxact:abstractionType>
<ipxact:abstractionType>
    <ipxact:viewRef>TLMview</ipxact:viewRef>
    <ipxact:abstractionRef vendor="accelera.org"
library="Sample" name="SampleAbstractionDefinition_TLM" version="1.0"/>
    <ipxact:portMaps>
        <ipxact:portMap>
            <ipxact:logicalPort>
                <ipxact:name>Transport</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
                <ipxact:name>init_transaction</ipxact:name>
            </ipxact:physicalPort>
        </ipxact:portMap>
    </ipxact:portMaps>
</ipxact:abstractionType>
</ipxact:abstractionTypes>
<!-- LINK: interfaceMode: see 6.7.4, Interface modes -->
<!-- LINK: initiator: see 6.7.5, Initiator interface -->
<ipxact:initiator>
    <ipxact:addressSpaceRef addressSpaceRef="simpleAddressSpace"/>
</ipxact:initiator>
</ipxact:busInterface>
<!-- Basic target interface with both RTL and TLM representations -->
<ipxact:busInterface>
    <ipxact:name>Target</ipxact:name>
    <ipxact:busType vendor="accelera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
    <ipxact:abstractionTypes>
        <ipxact:abstractionType>
            <ipxact:viewRef>RTLview</ipxact:viewRef>
            <ipxact:abstractionRef vendor="accelera.org"
library="Sample" name="SampleAbstractionDefinition_RTL" version="1.0">
                <!-- Configure abstraction to have data width 8 -->
                <ipxact:configurableElementValues>
                    <ipxact:configurableElementValue
referenceId="dataWidth">8</ipxact:configurableElementValue>
                </ipxact:configurableElementValues>
            </ipxact:abstractionRef>
        <ipxact:portMaps>
            <ipxact:portMap>
                <ipxact:logicalPort>
                    <ipxact:name>Data</ipxact:name>
                </ipxact:logicalPort>
                <ipxact:physicalPort>
                    <ipxact:name>target_data</ipxact:name>
                    <ipxact:partSelect>
                        <ipxact:range>
                            <ipxact:left>7</ipxact:left>
                            <ipxact:right>0</ipxact:right>
                        </ipxact:range>
                    </ipxact:partSelect>
                </ipxact:physicalPort>
            </ipxact:portMap>
        <ipxact:portMap>
    
```

```

<ipxact:logicalPort>
    <ipxact:name>Address</ipxact:name>
</ipxact:logicalPort>
<ipxact:physicalPort>
    <ipxact:name>target_addr</ipxact:name>
</ipxact:physicalPort>
</ipxact:portMap>
<!-- Mapping to clock port included for 'information' only
- not for connecting --&gt;
&lt;ipxact:portMap&gt;
    &lt;ipxact:logicalPort&gt;
        &lt;ipxact:name&gt;Clk&lt;/ipxact:name&gt;
    &lt;/ipxact:logicalPort&gt;
    &lt;ipxact:physicalPort&gt;
        &lt;ipxact:name&gt;clk&lt;/ipxact:name&gt;
    &lt;/ipxact:physicalPort&gt;
    &lt;ipxact:isInformative&gt;true&lt;/ipxact:isInformative&gt;
&lt;/ipxact:portMap&gt;
&lt;/ipxact:portMaps&gt;
&lt;/ipxact:abstractionType&gt;
&lt;ipxact:abstractionType&gt;
    &lt;ipxact:viewRef&gt;TLMview&lt;/ipxact:viewRef&gt;
    &lt;ipxact:abstractionRef vendor="accelera.org"
library="Sample" name="SampleAbstractionDefinition_TLM" version="1.0"/&gt;
    &lt;ipxact:portMaps&gt;
        &lt;ipxact:portMap&gt;
            &lt;ipxact:logicalPort&gt;
                &lt;ipxact:name&gt;Transport&lt;/ipxact:name&gt;
            &lt;/ipxact:logicalPort&gt;
            &lt;ipxact:physicalPort&gt;
                &lt;ipxact:name&gt;target_transaction&lt;/ipxact:name&gt;
            &lt;/ipxact:physicalPort&gt;
        &lt;/ipxact:portMap&gt;
    &lt;/ipxact:portMaps&gt;
&lt;/ipxact:abstractionType&gt;
&lt;/ipxact:abstractionTypes&gt;
<!-- LINK: target: see <a href="#">6.7.6, Target interface -->
<ipxact:target>
    <ipxact:memoryMapRef memoryMapRef="SimpleMapWithBlock"/>
</ipxact:target>
<!-- This interface must be connected in any containing design --&gt;
&lt;ipxact:connectionRequired&gt;true&lt;/ipxact:connectionRequired&gt;
&lt;/ipxact:busInterface&gt;
<!-- Interface with port maps of wire ports and structured port --&gt;
&lt;ipxact:busInterface&gt;
    &lt;ipxact:name&gt;APB&lt;/ipxact:name&gt;
    &lt;ipxact:busType vendor="amba.com" library="AMBA4" name="APB4"
version="r0p0_0"/&gt;
    &lt;ipxact:abstractionTypes&gt;
        &lt;ipxact:abstractionType&gt;
            &lt;ipxact:abstractionRef vendor="amba.com" library="AMBA4"
name="APB4_rtl" version="r0p0_0"/&gt;
            &lt;ipxact:portMaps&gt;
                &lt;ipxact:portMap&gt;
                    &lt;ipxact:logicalPort&gt;
                        &lt;ipxact:name&gt;PCLK&lt;/ipxact:name&gt;
                    &lt;/ipxact:logicalPort&gt;
                    &lt;ipxact:physicalPort&gt;
                        &lt;ipxact:name&gt;clk&lt;/ipxact:name&gt;
</pre>

```

```

        </ipxact:physicalPort>
    </ipxact:portMap>
    <ipxact:portMap>
        <ipxact:logicalPort>
            <ipxact:name>PRESETn</ipxact:name>
        </ipxact:logicalPort>
        <ipxact:physicalPort>
            <ipxact:name>reset</ipxact:name>
        </ipxact:physicalPort>
    </ipxact:portMap>
    <!-- Port map with physical port referencing a structured
port and subport -->
    <ipxact:portMap>
        <ipxact:logicalPort>
            <ipxact:name>PADDR</ipxact:name>
        </ipxact:logicalPort>
        <ipxact:physicalPort>
            <ipxact:name>apb</ipxact:name>
            <ipxact:subPort>
                <ipxact:name>addr</ipxact:name>
            </ipxact:subPort>
            </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
            <ipxact:logicalPort>
                <ipxact:name>PRDATA</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
                <ipxact:name>apb</ipxact:name>
                <ipxact:subPort>
                    <ipxact:name>rdata</ipxact:name>
                </ipxact:subPort>
            </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
            <ipxact:logicalPort>
                <ipxact:name>PWDATA</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
                <ipxact:name>apb</ipxact:name>
                <ipxact:subPort>
                    <ipxact:name>wdata</ipxact:name>
                </ipxact:subPort>
            </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
            <ipxact:logicalPort>
                <ipxact:name>PWRITE</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
                <ipxact:name>apb</ipxact:name>
                <ipxact:subPort>
                    <ipxact:name>write</ipxact:name>
                </ipxact:subPort>
            </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
            <ipxact:logicalPort>
                <ipxact:name>PENABLE</ipxact:name>

```

```

    </ipxact:logicalPort>
    <ipxact:physicalPort>
        <ipxact:name>apb</ipxact:name>
        <ipxact:subPort>
            <ipxact:name>enable</ipxact:name>
        </ipxact:subPort>
    </ipxact:physicalPort>
</ipxact:portMap>
<ipxact:portMap>
    <ipxact:logicalPort>
        <ipxact:name>PSELx</ipxact:name>
    </ipxact:logicalPort>
    <ipxact:physicalPort>
        <ipxact:name>apb</ipxact:name>
        <ipxact:subPort>
            <ipxact:name>sel</ipxact:name>
        </ipxact:subPort>
    </ipxact:physicalPort>
</ipxact:portMap>
<ipxact:portMap>
    <ipxact:logicalPort>
        <ipxact:name>PREADY</ipxact:name>
    </ipxact:logicalPort>
    <ipxact:physicalPort>
        <ipxact:name>apb</ipxact:name>
        <ipxact:subPort>
            <ipxact:name>ready</ipxact:name>
        </ipxact:subPort>
    </ipxact:physicalPort>
</ipxact:portMap>
<ipxact:portMap>
    <ipxact:logicalPort>
        <ipxact:name>PSLVERR</ipxact:name>
    </ipxact:logicalPort>
    <ipxact:physicalPort>
        <ipxact:name>apb</ipxact:name>
        <ipxact:subPort>
            <ipxact:name>slverr</ipxact:name>
        </ipxact:subPort>
    </ipxact:physicalPort>
</ipxact:portMap>
<ipxact:portMap>
    <ipxact:logicalPort>
        <ipxact:name>PPROT</ipxact:name>
    </ipxact:logicalPort>
    <ipxact:physicalPort>
        <ipxact:name>apb</ipxact:name>
        <ipxact:subPort>
            <ipxact:name>prot</ipxact:name>
        </ipxact:subPort>
    </ipxact:physicalPort>
</ipxact:portMap>
<ipxact:portMap>
    <ipxact:logicalPort>
        <ipxact:name>PSTROBE</ipxact:name>
    </ipxact:logicalPort>
    <ipxact:physicalPort>
        <ipxact:name>apb</ipxact:name>
        <ipxact:subPort>

```

```

        <ipxact:name>strb</ipxact:name>
        </ipxact:subPort>
        </ipxact:physicalPort>
        </ipxact:portMap>
        </ipxact:portMaps>
        </ipxact:abstractionType>
        </ipxact:abstractionTypes>
        <ipxact:target/>
    </ipxact:busInterface>
    <ipxact:busInterface>
        <ipxact:name>MirroredInitiator</ipxact:name>
        <ipxact:busType vendor="accelera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
        <ipxact:mirroredInitiator/>
    </ipxact:busInterface>
    <ipxact:busInterface>
        <ipxact:name>MirroredTarget</ipxact:name>
        <ipxact:busType vendor="accelera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
        <!-- LINK: mirroredTarget: see 6.7.7, Mirrored target interface -->
        <ipxact:mirroredTarget>
            <ipxact:baseAddresses>
                <ipxact:remapAddress>'h40000000</ipxact:remapAddress>
                <ipxact:range>'h1000</ipxact:range>
            </ipxact:baseAddresses>
        </ipxact:mirroredTarget>
    </ipxact:busInterface>
    <ipxact:busInterface>
        <ipxact:name>TargetWithTransparentBridge</ipxact:name>
        <ipxact:busType vendor="accelera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
        <ipxact:target>
            <ipxact:transparentBridge initiatorRef="Initiator"/>
        </ipxact:target>
    </ipxact:busInterface>
    <ipxact:busInterface>
        <ipxact:name>TargetForSubspaceMap</ipxact:name>
        <ipxact:busType vendor="accelera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
        <ipxact:target>
            <ipxact:memoryMapRef memoryMapRef="SimpleMapWithSubspace"/>
        </ipxact:target>
    </ipxact:busInterface>
    <ipxact:busInterface>
        <ipxact:name>Monitor</ipxact:name>
        <ipxact:busType vendor="accelera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
        <ipxact:monitor interfaceMode="target"/>
    </ipxact:busInterface>
</ipxact:busInterfaces>
<!-- LINK: indirectInterface: see 6.8, Indirect interfaces -->
<ipxact:indirectInterfaces>
    <ipxact:indirectInterface>
        <ipxact:name>IndirectMemoryInterface</ipxact:name>
        <ipxact:indirectAddressRef>
            <ipxact:memoryMapRef memoryMapRef="SimpleMapWithBlock"/>
            <ipxact:addressBlockRef addressBlockRef="SimpleAddressBlock"/>
            <ipxact:registerRef registerRef="IAR"/>
            <ipxact:fieldRef fieldRef="IAR"/>

```

```

        </ipxact:indirectAddressRef>
        <ipxact:indirectDataRef>
            <ipxact:memoryMapRef memoryMapRef="SimpleMapWithBlock"/>
            <ipxact:addressBlockRef addressBlockRef="SimpleAddressBlock"/>
            <ipxact:registerRef registerRef="IDR"/>
            <ipxact:fieldRef fieldRef="IDR"/>
        </ipxact:indirectDataRef>
        <ipxact:memoryMapRef>MapForMemory</ipxact:memoryMapRef>
    </ipxact:indirectInterface>
</ipxact:indirectInterfaces>
<!-- LINK: channels: see 6.9, Component channels -->
<ipxact:channels>
    <ipxact:channel>
        <ipxact:name>theChannel</ipxact:name>
        <ipxact:busInterfaceRef>
            <ipxact:localName>MirroredInitiator</ipxact:localName>
        </ipxact:busInterfaceRef>
        <ipxact:busInterfaceRef>
            <ipxact:localName>MirroredTarget</ipxact:localName>
        </ipxact:busInterfaceRef>
    </ipxact:channel>
</ipxact:channels>
<!-- LINK: modes: see 6.10, Modes -->
<ipxact:modes>
    <ipxact:mode>
        <ipxact:name>Boot</ipxact:name>
        <ipxact:description>Defines the remap state associated with the
component at boot time.</ipxact:description>
    </ipxact:mode>
    <ipxact:mode>
        <ipxact:name>Normal</ipxact:name>
        <ipxact:description>Defines the remap state associated with the
component after boot time.</ipxact:description>
    </ipxact:mode>
    <ipxact:mode>
        <ipxact:name>AlternateRegisterGroup</ipxact:name>
        <ipxact:description>Defines the state associated with alternate
registers in this component.</ipxact:description>
    </ipxact:mode>
</ipxact:modes>
<!-- LINK: addressSpaces: see 6.11, Address spaces -->
<ipxact:addressSpaces>
    <ipxact:addressSpace>
        <ipxact:name>simpleAddressSpace</ipxact:name>
        <ipxact:range>4*(2**30)</ipxact:range>
        <ipxact:width>32</ipxact:width>
    </ipxact:addressSpace>
</ipxact:addressSpaces>
<!-- LINK: memoryMaps: see 6.12, Memory maps -->
<ipxact:memoryMaps>
    <ipxact:memoryMap>
        <ipxact:name>SimpleMapWithBlock</ipxact:name>
        <!-- Address block starts at address 0, containing 1024 addressable
8-bit -->
        <!-- units, organized into larger 32-bit units. -->
        <!-- LINK: addressBlock: see 6.12.2, Address block -->
        <ipxact:addressBlock>
            <ipxact:name>SimpleAddressBlock</ipxact:name>
            <ipxact:baseAddress>'h0</ipxact:baseAddress>

```

```

    <!-- LINK: addressBlockDefinitionGroup: see 6.12.3, Address
    block definition group -->
    <ipxact:range>2**10</ipxact:range>
    <ipxact:width>32</ipxact:width>
    <!-- LINK: memoryBlockData: see 6.12.4, memoryBlockData group -->
    <ipxact:usage>register</ipxact:usage>
    <ipxact:volatile>false</ipxact:volatile>
    <ipxact:accessPolicies>
        <ipxact:accessPolicy>
            <ipxact:access>read-write</ipxact:access>
        </ipxact:accessPolicy>
    </ipxact:accessPolicies>
    <!-- LINK: registerData: see 6.14.1, Register data -->
    <!-- LINK: register: see 6.14.2, Register -->
    <ipxact:register>
        <ipxact:name>BasicRegister</ipxact:name>
        <ipxact:addressOffset>'h4</ipxact:addressOffset>
        <!-- LINK: registerDefinitionGroup: see 6.14.3, Register
        definition group -->
        <ipxact:size>32</ipxact:size>
        <ipxact:volatile>true</ipxact:volatile>
        <ipxact:accessPolicies>
            <ipxact:accessPolicy>
                <ipxact:access>read-writeOnce</ipxact:access>
            </ipxact:accessPolicy>
        </ipxact:accessPolicies>
        <!-- LINK: field: see 6.14.8, Register bit fields -->
        <ipxact:field>
            <ipxact:name>F1</ipxact:name>
            <ipxact:bitOffset>0</ipxact:bitOffset>
            <ipxact:bitWidth>4</ipxact:bitWidth>
            <ipxact:resets>
                <ipxact:reset>
                    <ipxact:value>'h0</ipxact:value>
                </ipxact:reset>
                <ipxact:reset resetTypeRef="SOFT">
                    <ipxact:value>'hf</ipxact:value>
                    <ipxact:mask>'ha</ipxact:mask>
                </ipxact:reset>
            </ipxact:resets>
            <!-- LINK: fieldDataGroup: see 6.14.9, Field data group ->
        </ipxact:field>
        <ipxact:fieldAccessPolicies>
            <ipxact:fieldAccessPolicy>
                <ipxact:access>writeOnce</ipxact:access>
            </ipxact:fieldAccessPolicy>
        </ipxact:fieldAccessPolicies>
        <ipxact:enumeratedValues>
            <!-- LINK: enumeratedValue: see 6.14.12, Enumeration
            values -->
            <ipxact:enumeratedValue>
                <ipxact:name>SetPos0</ipxact:name>
                <ipxact:value>'h1</ipxact:value>
            </ipxact:enumeratedValue>
            <ipxact:enumeratedValue>
                <ipxact:name>SetPos1</ipxact:name>
                <ipxact:value>'h2</ipxact:value>
            </ipxact:enumeratedValue>
        </ipxact:enumeratedValues>
    </ipxact:register>

```

```

        </ipxact:field>
        <ipxact:field>
            <ipxact:name>F2</ipxact:name>
            <ipxact:bitOffset>4</ipxact:bitOffset>
            <ipxact:bitWidth>4</ipxact:bitWidth>
            <ipxact:resets>
                <ipxact:reset>
                    <ipxact:value>'h0</ipxact:value>
                </ipxact:reset>
            </ipxact:resets>
            <ipxact:fieldAccessPolicies>
                <ipxact:fieldAccessPolicy>
                    <ipxact:modifiedWriteValue>oneToClear</
ipxact:modifiedWriteValue>
                    <ipxact:readAction>modify</ipxact:readAction>
                    <ipxact:testable testConstraint="readOnly">true</
ipxact:testable>
                </ipxact:fieldAccessPolicy>
            </ipxact:fieldAccessPolicies>
        </ipxact:field>
        <ipxact:field>
            <ipxact:name>F3</ipxact:name>
            <ipxact:bitOffset>8</ipxact:bitOffset>
            <ipxact:aliasOf>
                <ipxact:fieldRef fieldRef="F2"/>
            </ipxact:aliasOf>
            <ipxact:fieldAccessPolicies>
                <ipxact:fieldAccessPolicy>
                    <ipxact:access>read-only</ipxact:access>
                </ipxact:fieldAccessPolicy>
            </ipxact:fieldAccessPolicies>
        </ipxact:field>
        <ipxact:field>
            <ipxact:name>F4</ipxact:name>
            <ipxact:bitOffset>12</ipxact:bitOffset>
            <ipxact:bitWidth>4</ipxact:bitWidth>
            <ipxact:fieldAccessPolicies>
                <ipxact:fieldAccessPolicy>
                    <!-- LINK: writeValueConstraint: see 6.14.9, Field
data group -->
                    <ipxact:writeValueConstraint>
                        <ipxact:minimum>'h1</ipxact:minimum>
                        <ipxact:maximum>'h2</ipxact:maximum>
                    </ipxact:writeValueConstraint>
                </ipxact:fieldAccessPolicy>
            </ipxact:fieldAccessPolicies>
        </ipxact:field>
        <ipxact:field>
            <ipxact:name>F5</ipxact:name>
            <ipxact:bitOffset>16</ipxact:bitOffset>
            <ipxact:bitWidth>16</ipxact:bitWidth>
            <ipxact:fieldAccessPolicies>
                <ipxact:fieldAccessPolicy>
                    <ipxact:reserved>true</ipxact:reserved>
                </ipxact:fieldAccessPolicy>
            </ipxact:fieldAccessPolicies>
        </ipxact:field>
    </ipxact:register>
    <ipxact:register>

```

```

<ipxact:name>IAR</ipxact:name>
<ipxact:addressOffset>'h8</ipxact:addressOffset>
<ipxact:size>32</ipxact:size>
<ipxact:field>
    <ipxact:name>IAR</ipxact:name>
    <ipxact:bitOffset>0</ipxact:bitOffset>
    <ipxact:bitWidth>32</ipxact:bitWidth>
    <ipxact:fieldAccessPolicies>
        <ipxact:fieldAccessPolicy>
            <ipxact:broadcasts>
                <ipxact:broadcastTo>
                    <ipxact:memoryMapRef
memoryMapRef="SimpleMapWithBlock"/>
                    <ipxact:addressBlockRef
addressBlockRef="SimpleAddressBlock"/>
                    <ipxact:registerRef
registerRef="RegisterWithAlternate"/>
                    <ipxact:fieldRef fieldRef="F"/>
                </ipxact:broadcastTo>
            </ipxact:broadcasts>
        </ipxact:fieldAccessPolicy>
    </ipxact:fieldAccessPolicies>
    </ipxact:field>
</ipxact:register>
<ipxact:register>
    <ipxact:name>IDR</ipxact:name>
    <ipxact:addressOffset>'hc</ipxact:addressOffset>
    <ipxact:size>32</ipxact:size>
    <ipxact:field>
        <ipxact:name>IDR</ipxact:name>
        <ipxact:bitOffset>0</ipxact:bitOffset>
        <ipxact:bitWidth>32</ipxact:bitWidth>
    </ipxact:field>
</ipxact:register>
<ipxact:register>
    <ipxact:name>RegisterWithAlternate</ipxact:name>
    <ipxact:addressOffset>'h10</ipxact:addressOffset>
    <ipxact:size>32</ipxact:size>
    <ipxact:accessPolicies>
        <ipxact:accessPolicy>
            <ipxact:access>write-only</ipxact:access>
        </ipxact:accessPolicy>
    </ipxact:accessPolicies>
    <ipxact:field>
        <ipxact:name>F</ipxact:name>
        <ipxact:bitOffset>0</ipxact:bitOffset>
        <ipxact:bitWidth>32</ipxact:bitWidth>
    </ipxact:field>
    <ipxact:alternateRegisters>
        <!-- LINK: alternateRegister: see 6.14.4, Alternate
register -->
        <ipxact:alternateRegister>
            <ipxact:name>Alternate1</ipxact:name>
            <ipxact:modeRef priority="1">AlternateRegisterGroup</
ipxact:modeRef>
            <!-- LINK: alternateRegisterDefinitionGroup: see
6.14.7, Alternate register definition group -->
            <ipxact:accessPolicies>
                <ipxact:accessPolicy>

```

```

        <ipxact:access>read-only</ipxact:access>
        </ipxact:accessPolicy>
        </ipxact:accessPolicies>
        <ipxact:field>
            <ipxact:name>F</ipxact:name>
            <ipxact:bitOffset>0</ipxact:bitOffset>
            <ipxact:bitWidth>32</ipxact:bitWidth>
        </ipxact:field>
        </ipxact:alternateRegister>
        </ipxact:alternateRegisters>
    </ipxact:register>
    <!-- 2 registers combined in a register file array with 32 bit
gap -->
    <!-- LINK: registerFile: see 6.14.6, Register file -->
    <ipxact:registerFile>
        <ipxact:name>RegisterFile</ipxact:name>
        <ipxact:array>
            <!-- dim element with indexVar attribute -->
            <ipxact:dim indexVar="i">8</ipxact:dim>
            <!-- registerFile elements offset are 'h10 address unit
bits apart -->
            <ipxact:stride>'h10</ipxact:stride>
        </ipxact:array>
        <ipxact:addressOffset>'h200</ipxact:addressOffset>
        <ipxact:range>'h8</ipxact:range>
        <ipxact:register>
            <ipxact:name>RegFileEntry1</ipxact:name>
            <!-- Description using escape sequence
$ipxact_index_value() -->
            <ipxact:description>Description for RegFileEntry1 element
$ipxact_index_value(i)</ipxact:description>
            <ipxact:addressOffset>'h0</ipxact:addressOffset>
            <ipxact:size>32</ipxact:size>
            <ipxact:field>
                <ipxact:name>MandatoryField</ipxact:name>
                <ipxact:bitOffset>0</ipxact:bitOffset>
                <ipxact:bitWidth>32</ipxact:bitWidth>
            </ipxact:field>
        </ipxact:register>
        <ipxact:register>
            <ipxact:name>RegFileEntry2</ipxact:name>
            <ipxact:addressOffset>'h4</ipxact:addressOffset>
            <ipxact:size>32</ipxact:size>
            <ipxact:field>
                <ipxact:name>MandatoryField</ipxact:name>
                <ipxact:bitOffset>0</ipxact:bitOffset>
                <ipxact:bitWidth>32</ipxact:bitWidth>
            </ipxact:field>
        </ipxact:register>
    </ipxact:registerFile>
    </ipxact:addressBlock>
    <ipxact:addressUnitBits>8</ipxact:addressUnitBits>
</ipxact:memoryMap>
<ipxact:memoryMap>
    <ipxact:name>SimpleMapWithBank</ipxact:name>
    <!-- Serial bank with two memory blocks of 1 k units of 32-bit data.
-->
    <!-- The only address specified is 'h10000, but this causes address
block -->

```

```

    <!-- ram0 and ram1 to be mapped to addresses 'h10000 and 'h11000
    respectively. -->
    <!-- LINK: addressBank: see 6.12.5, Bank -->
    <ipxact:bank bankAlignment="serial">
        <ipxact:name>SerialBank</ipxact:name>
        <ipxact:baseAddress>'h1000</ipxact:baseAddress>
        <ipxact:addressBlock>
            <ipxact:name>ram0</ipxact:name>
            <ipxact:range>'h1000</ipxact:range>
            <ipxact:width>32</ipxact:width>
        </ipxact:addressBlock>
        <ipxact:addressBlock>
            <ipxact:name>ram1</ipxact:name>
            <ipxact:range>'h1000</ipxact:range>
            <ipxact:width>32</ipxact:width>
        </ipxact:addressBlock>
    </ipxact:bank>
</ipxact:memoryMap>
<ipxact:memoryMap>
    <ipxact:name>SimpleMapWithSubspace</ipxact:name>
    <!-- Address space from initiator interface 'Initiator' mapped into
target interface -->
    <!-- TargetForSubspaceMap at address 'h10000 -->
    <!-- LINK: subspaceMap: see 6.12.9, Subspace map -->
    <ipxact:subspaceMap initiatorRef="Initiator">
        <ipxact:name>SubSpace</ipxact:name>
        <ipxact:baseAddress>'h10000</ipxact:baseAddress>
    </ipxact:subspaceMap>
</ipxact:memoryMap>
<ipxact:memoryMap>
    <ipxact:name>MapForMemory</ipxact:name>
    <ipxact:addressBlock>
        <ipxact:name>MemoryBlock</ipxact:name>
        <ipxact:baseAddress>'h0</ipxact:baseAddress>
        <ipxact:range>2**10</ipxact:range>
        <ipxact:width>32</ipxact:width>
        <ipxact:usage>memory</ipxact:usage>
        <ipxact:accessPolicies>
            <ipxact:accessPolicy>
                <ipxact:access>read-write</ipxact:access>
            </ipxact:accessPolicy>
        </ipxact:accessPolicies>
    </ipxact:addressBlock>
</ipxact:memoryMap>
<ipxact:memoryMap>
    <ipxact:name>cpuDefn_MemoryMap</ipxact:name>
    <ipxact:subspaceMap initiatorRef="Initiator">
        <ipxact:name>simpleAddressSpace_SubspaceMap</ipxact:name>
        <ipxact:baseAddress>0</ipxact:baseAddress>
    </ipxact:subspaceMap>
</ipxact:memoryMap>
</ipxact:memoryMaps>
<!-- LINK: model: see 6.15, Model -->
<ipxact:model>
    <ipxact:views>
        <ipxact:view>
            <ipxact:name>RTLview</ipxact:name>
            <ipxact:displayName>RTL View</ipxact:displayName>

```

```

        <ipxact:description>Simple RTL view of a component.</
    ipxact:description>
        <ipxact:envIdentifier>*:Synthesis:</ipxact:envIdentifier>
        <ipxact:componentInstantiationRef>VerilogModel</
    ipxact:componentInstantiationRef>
        </ipxact:view>
    <ipxact:view>
        <ipxact:name>TLMview</ipxact:name>
        <ipxact:displayName>TLM View</ipxact:displayName>
        <ipxact:description>Simple TLM view of a component.</
    ipxact:description>
        <ipxact:envIdentifier>*:Simulation:</ipxact:envIdentifier>
        <ipxact:componentInstantiationRef>TLMMModel</
    ipxact:componentInstantiationRef>
        </ipxact:view>
    </ipxact:views>
    <ipxact:instantiations>
        <!-- LINK: instantiationsGroup: see 6.15.2, instantiationsGroup -->
        <!-- LINK: componentInstantiation: see 6.15.3, componentInstantiation -->
        <ipxact:componentInstantiation>
            <ipxact:componentInstantiation>
                <ipxact:name>VerilogModel</ipxact:name>
                <ipxact:language>verilog</ipxact:language>
                <ipxact:moduleName>sample</ipxact:moduleName>
                <!-- LINK: moduleParameters: see 6.15.6, Module parameters -->
                <ipxact:moduleParameters>
                    <ipxact:moduleParameter type="bit">
                        <ipxact:name>dual_mode_reg_value</ipxact:name>
                        <ipxact:vectors>
                            <ipxact:vector>
                                <ipxact:left>3</ipxact:left>
                                <ipxact:right>0</ipxact:right>
                            </ipxact:vector>
                        </ipxact:vectors>
                        <ipxact:value>comp_dual_mode?4'hf:4'h0</ipxact:value>
                    </ipxact:moduleParameter>
                </ipxact:moduleParameters>
                <ipxact:fileSetRef>
                    <ipxact:localName>VerilogFiles</ipxact:localName>
                </ipxact:fileSetRef>
            </ipxact:componentInstantiation>
            <ipxact:componentInstantiation>
                <ipxact:name>TLMMModel</ipxact:name>
                <ipxact:language>SystemC</ipxact:language>
                <ipxact:moduleName>sample</ipxact:moduleName>
                <ipxact:moduleParameters>
                    <ipxact:moduleParameter type="bit">
                        <ipxact:name>dual_mode</ipxact:name>
                        <ipxact:value>comp_dual_mode</ipxact:value>
                    </ipxact:moduleParameter>
                    <ipxact:moduleParameter parameterId="vdp" resolve="generated" type="bit">
                        <ipxact:name>view_dependent_param</ipxact:name>
                        <ipxact:value>0</ipxact:value>
                    </ipxact:moduleParameter>
                </ipxact:moduleParameters>
                <ipxact:fileSetRef>
                    <ipxact:localName>SystemCFiles</ipxact:localName>
                </ipxact:fileSetRef>
            </ipxact:componentInstantiation>
        </ipxact:instantiations>
    </ipxact:componentInstantiation>

```

```

        </ipxact:componentInstantiation>
    </ipxact:instantiations>
    <!-- LINK: port: see 6.15.7, Component ports -->
    <ipxact:ports>
        <ipxact:port>
            <ipxact:name>target_data</ipxact:name>
            <!-- LINK: wire: see 6.15.8, Component wire ports -->
            <ipxact:wire>
                <ipxact:direction>in</ipxact:direction>
                <ipxact:vectors>
                    <ipxact:vector>
                        <ipxact:left>15</ipxact:left>
                        <ipxact:right>0</ipxact:right>
                    </ipxact:vector>
                </ipxact:vectors>
                <ipxact:wireTypeDefs>
                    <!-- LINK: wireTypeDef: see 6.15.11, Component wireTypeDef -->
                </ipxact:wireTypeDefs>
                <!-- Default upper 8 bits since the interface only uses 8 bits -->
            </ipxact:wire>
        </ipxact:port>
        <ipxact:port>
            <ipxact:name>init_data</ipxact:name>
            <ipxact:wire>
                <ipxact:direction>out</ipxact:direction>
                <ipxact:vectors>
                    <ipxact:vector>
                        <ipxact:left>7</ipxact:left>
                        <ipxact:right>0</ipxact:right>
                    </ipxact:vector>
                </ipxact:vectors>
                <ipxact:wireTypeDefs>
                    <ipxact:wireTypeDef>
                        <ipxact:typeName>logic</ipxact:typeName>
                        <ipxact:viewRef>RTLview</ipxact:viewRef>
                    </ipxact:wireTypeDef>
                </ipxact:wireTypeDefs>
                <ipxact:powerConstraints>

```

```

        <ipxact:powerConstraint>
            <ipxact:powerDomainRef>PD2</ipxact:powerDomainRef>
        </ipxact:powerConstraint>
    </ipxact:powerConstraints>
</ipxact:wire>
</ipxact:port>
<ipxact:port>
    <ipxact:name>target_addr</ipxact:name>
    <ipxact:wire>
        <ipxact:direction>in</ipxact:direction>
        <ipxact:vectors>
            <ipxact:vector>
                <ipxact:left>7</ipxact:left>
                <ipxact:right>0</ipxact:right>
            </ipxact:vector>
        </ipxact:vectors>
        <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
                <ipxact:typeName>logic</ipxact:typeName>
                <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
        <ipxact:constraintSets>
            <!-- LINK: constraintSet: see 6.15.15, Component wire port
constraints -->
            <ipxact:constraintSet>
                <!-- Port driven by high strength sequential cell for
synthesis -->
                <ipxact:driveConstraint>
                    <!-- LINK: cellSpecification: see 6.15.17, Load and
drive constraint cell specification -->
                    <ipxact:cellSpecification cellStrength="high">
                        <ipxact:cellClass>sequential</
ipxact:cellClass>
                    </ipxact:cellSpecification>
                </ipxact:driveConstraint>
                <!-- Port timing requirements for synthesis -->
                <!-- LINK: timingConstraint: see 6.15.16, Port timing
constraints -->
                <ipxact:timingConstraint clockName="clk">60</
ipxact:timingConstraint>
                </ipxact:constraintSet>
            </ipxact:constraintSets>
            <ipxact:powerConstraints>
                <ipxact:powerConstraint>
                    <ipxact:powerDomainRef>PD1</ipxact:powerDomainRef>
                </ipxact:powerConstraint>
            </ipxact:powerConstraints>
        </ipxact:wire>
    </ipxact:port>
    <ipxact:port>
        <ipxact:name>init_addr</ipxact:name>
        <ipxact:wire>
            <ipxact:direction>out</ipxact:direction>
            <ipxact:vectors>
                <ipxact:vector>
                    <ipxact:left>7</ipxact:left>
                    <ipxact:right>0</ipxact:right>
                </ipxact:vector>
            </ipxact:vectors>
        </ipxact:wire>
    </ipxact:port>

```

```

        </ipxact:vectors>
        <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
                <ipxact:typeName>logic</ipxact:typeName>
                <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
        <ipxact:constraintSets>
            <ipxact:constraintSet>
                <!-- Port drives 2 high strength sequential cells for
synthesis -->
                <ipxact:loadConstraint>
                    <ipxact:cellSpecification cellStrength="high">
                        <ipxact:cellClass>sequential</
ipxact:cellClass>
                    </ipxact:cellSpecification>
                    <ipxact:count>2</ipxact:count>
                </ipxact:loadConstraint>
                <!-- Port timing requirements for synthesis -->
                <ipxact:timingConstraint clockName="clk">40</
ipxact:timingConstraint>
                <ipxact:constraintSet>
            </ipxact:constraintSets>
            <ipxact:powerConstraints>
                <ipxact:powerConstraint>
                    <ipxact:powerDomainRef>PD2</ipxact:powerDomainRef>
                </ipxact:powerConstraint>
            </ipxact:powerConstraints>
        </ipxact:wire>
        </ipxact:port>
        <!-- parity port exists in RTL view with default data type -->
        <ipxact:port>
            <ipxact:name>target_parity</ipxact:name>
            <ipxact:wire>
                <ipxact:direction>in</ipxact:direction>
                <ipxact:wireTypeDefs>
                    <ipxact:wireTypeDef>
                        <ipxact:viewRef>RTLview</ipxact:viewRef>
                    </ipxact:wireTypeDef>
                </ipxact:wireTypeDefs>
                <ipxact:powerConstraints>
                    <ipxact:powerConstraint>
                        <ipxact:powerDomainRef>PD1</ipxact:powerDomainRef>
                    </ipxact:powerConstraint>
                </ipxact:powerConstraints>
            </ipxact:wire>
        </ipxact:port>
        <!-- clock port exists in RTL view with default data type -->
        <ipxact:port>
            <ipxact:name>clk</ipxact:name>
            <ipxact:wire>
                <ipxact:direction>in</ipxact:direction>
                <ipxact:wireTypeDefs>
                    <ipxact:wireTypeDef>
                        <ipxact:viewRef>RTLview</ipxact:viewRef>
                    </ipxact:wireTypeDef>
                </ipxact:wireTypeDefs>
                <ipxact:drivers>
                    <ipxact:driver>

```

```

        <!-- LINK: clockDriver: see 6.15.13, Component driver/
clockDriver -->
        <ipxact:clockDriver>
            <ipxact:clockPeriod>5</ipxact:clockPeriod>
            <ipxact:clockPulseOffset>2.5</
ipxact:clockPulseOffset>
            <ipxact:clockPulseValue>1</ipxact:clockPulseValue>
            <ipxact:clockPulseDuration>2.5</
ipxact:clockPulseDuration>
        </ipxact:clockDriver>
        </ipxact:driver>
    </ipxact:drivers>
    <ipxact:powerConstraints>
        <ipxact:powerConstraint>
            <ipxact:powerDomainRef>PD1</ipxact:powerDomainRef>
        </ipxact:powerConstraint>
    </ipxact:powerConstraints>
    </ipxact:wire>
</ipxact:port>
<!-- reset port exists in RTL view with default data type -->
<ipxact:port>
    <ipxact:name>reset</ipxact:name>
    <ipxact:wire>
        <ipxact:direction>in</ipxact:direction>
        <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
                <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
        <ipxact:drivers>
            <ipxact:driver>
                <ipxact:singleShotDriver>
                    <ipxact:singleShotOffset>2</
ipxact:singleShotOffset>
                    <ipxact:singleShotValue>1</ipxact:singleShotValue>
                    <ipxact:singleShotDuration>5</
ipxact:singleShotDuration>
                </ipxact:singleShotDriver>
            </ipxact:driver>
        </ipxact:drivers>
        <ipxact:powerConstraints>
            <ipxact:powerConstraint>
                <ipxact:powerDomainRef>PD1</ipxact:powerDomainRef>
            </ipxact:powerConstraint>
        </ipxact:powerConstraints>
    </ipxact:wire>
</ipxact:port>
<!-- status port exists in all views with default data type -->
<ipxact:port>
    <ipxact:name>status</ipxact:name>
    <ipxact:wire>
        <ipxact:direction>out</ipxact:direction>
        <ipxact:vectors>
            <ipxact:vector>
                <ipxact:left>7</ipxact:left>
                <ipxact:right>0</ipxact:right>
            </ipxact:vector>
        </ipxact:vectors>
        <ipxact:powerConstraints>

```

```

        <ipxact:powerConstraint>
            <ipxact:powerDomainRef>PD2</ipxact:powerDomainRef>
        </ipxact:powerConstraint>
    </ipxact:powerConstraints>
</ipxact:wire>
</ipxact:port>
<!-- anotherPort port exists in all views with default data type --&gt;
&lt;ipxact:port&gt;
    &lt;ipxact:name&gt;anotherPort&lt;/ipxact:name&gt;
    &lt;ipxact:wire&gt;
        &lt;ipxact:direction&gt;in&lt;/ipxact:direction&gt;
        &lt;ipxact:vectors&gt;
            &lt;ipxact:vector&gt;
                &lt;ipxact:left&gt;3&lt;/ipxact:left&gt;
                &lt;ipxact:right&gt;0&lt;/ipxact:right&gt;
            &lt;/ipxact:vector&gt;
        &lt;/ipxact:vectors&gt;
        &lt;ipxact:powerConstraints&gt;
            &lt;ipxact:powerConstraint&gt;
                &lt;ipxact:powerDomainRef&gt;PD1&lt;/ipxact:powerDomainRef&gt;
            &lt;/ipxact:powerConstraint&gt;
        &lt;/ipxact:powerConstraints&gt;
    &lt;/ipxact:wire&gt;
&lt;/ipxact:port&gt;
<!-- target_transaction port exists in TLM view only --&gt;
&lt;ipxact:port&gt;
    &lt;ipxact:name&gt;target_transaction&lt;/ipxact:name&gt;
    <!-- LINK: transactional: see <a href="#">6.15.19, Component transactional
port type -->
    <ipxact:transactional>
        <ipxact:initiative>requires</ipxact:initiative>
        <ipxact:transTypeDefs>
            <!-- LINK: transTypeDef: see <a href="#">6.15.21, Component
transactional port type definition -->
            <ipxact:transTypeDef>
                <ipxact:typeName>sampleTLMtype</ipxact:typeName>
                <ipxact:viewRef>TLMview</ipxact:viewRef>
            </ipxact:transTypeDef>
        </ipxact:transTypeDefs>
        <ipxact:powerConstraints>
            <ipxact:powerConstraint>
                <ipxact:powerDomainRef>PD1</ipxact:powerDomainRef>
            </ipxact:powerConstraint>
        </ipxact:powerConstraints>
    </ipxact:transactional>
</ipxact:port>
<!-- init_transaction port exists in TLM view only --&gt;
&lt;ipxact:port&gt;
    &lt;ipxact:name&gt;init_transaction&lt;/ipxact:name&gt;
    &lt;ipxact:transactional&gt;
        &lt;ipxact:initiative&gt;provides&lt;/ipxact:initiative&gt;
        &lt;ipxact:transTypeDefs&gt;
            &lt;ipxact:transTypeDef&gt;
                &lt;ipxact:typeName&gt;sampleTLMtype&lt;/ipxact:typeName&gt;
                &lt;ipxact:viewRef&gt;TLMview&lt;/ipxact:viewRef&gt;
            &lt;/ipxact:transTypeDef&gt;
        &lt;/ipxact:transTypeDefs&gt;
        &lt;ipxact:powerConstraints&gt;
            &lt;ipxact:powerConstraint&gt;
</pre>

```

```

        <ipxact:powerDomainRef>PD2</ipxact:powerDomainRef>
        </ipxact:powerConstraint>
        </ipxact:powerConstraints>
        </ipxact:transactional>
    </ipxact:port>
    <!-- apb structured port -->
    <ipxact:port>
        <ipxact:name>apb</ipxact:name>
        <!-- Link: structured: see 6.15.21, Component structured port
type -->
        <ipxact:structured>
            <ipxact:interface/>
            <!-- Link: subPorts: see 6.15.22, Subports -->
            <ipxact:subPorts>
                <ipxact:subPort>
                    <ipxact:name>addr</ipxact:name>
                    <ipxact:wire>
                        <ipxact:direction>in</ipxact:direction>
                        <ipxact:vectors>
                            <ipxact:vector>
                                <ipxact:left>clog2 (RANGE_id)-1</
ipxact:left>
                                <ipxact:right>0</ipxact:right>
                            </ipxact:vector>
                        </ipxact:vectors>
                    <ipxact:powerConstraints>
                        <ipxact:powerConstraint>
                            <ipxact:powerDomainRef>PD1</
ipxact:powerDomainRef>
                        </ipxact:powerConstraint>
                    </ipxact:powerConstraints>
                </ipxact:subPort>
                <ipxact:subPort>
                    <ipxact:name>prot</ipxact:name>
                    <ipxact:wire>
                        <ipxact:direction>in</ipxact:direction>
                        <ipxact:vectors>
                            <ipxact:vector>
                                <ipxact:left>2</ipxact:left>
                                <ipxact:right>0</ipxact:right>
                            </ipxact:vector>
                        </ipxact:vectors>
                    <ipxact:powerConstraints>
                        <ipxact:powerConstraint>
                            <ipxact:powerDomainRef>PD1</
ipxact:powerDomainRef>
                        </ipxact:powerConstraint>
                    </ipxact:powerConstraints>
                </ipxact:subPort>
                <ipxact:subPort>
                    <ipxact:name>sel</ipxact:name>
                    <ipxact:wire>
                        <ipxact:direction>in</ipxact:direction>
                    <ipxact:powerConstraints>
                        <ipxact:powerConstraint>
                            <ipxact:powerDomainRef>PD1</
ipxact:powerDomainRef>
                        </ipxact:powerConstraint>
                    </ipxact:powerConstraints>
                </ipxact:subPort>
            </ipxact:subPorts>
        </ipxact:structured>
    </ipxact:port>

```

```

        </ipxact:powerConstraint>
        </ipxact:powerConstraints>
        </ipxact:wire>
    </ipxact:subPort>
    <ipxact:subPort>
        <ipxact:name>enable</ipxact:name>
        <ipxact:wire>
            <ipxact:direction>in</ipxact:direction>
            <ipxact:powerConstraints>
                <ipxact:powerConstraint>
                    <ipxact:powerDomainRef>PD1</
                    ipxact:powerDomainRef>
                    </ipxact:powerConstraint>
                    </ipxact:powerConstraints>
                </ipxact:wire>
            </ipxact:subPort>
            <ipxact:subPort>
                <ipxact:name>write</ipxact:name>
                <ipxact:wire>
                    <ipxact:direction>in</ipxact:direction>
                    <ipxact:powerConstraints>
                        <ipxact:powerConstraint>
                            <ipxact:powerDomainRef>PD1</
                            ipxact:powerDomainRef>
                            </ipxact:powerConstraint>
                            </ipxact:powerConstraints>
                        </ipxact:wire>
                    </ipxact:subPort>
                    <ipxact:subPort>
                        <ipxact:name>wdata</ipxact:name>
                        <ipxact:wire>
                            <ipxact:direction>in</ipxact:direction>
                            <ipxact:vectors>
                                <ipxact:vector>
                                    <ipxact:left>31</ipxact:left>
                                    <ipxact:right>0</ipxact:right>
                                </ipxact:vector>
                            </ipxact:vectors>
                            <ipxact:powerConstraints>
                                <ipxact:powerConstraint>
                                    <ipxact:powerDomainRef>PD1</
                                    ipxact:powerDomainRef>
                                    </ipxact:powerConstraint>
                                    </ipxact:powerConstraints>
                                </ipxact:wire>
                            </ipxact:subPort>
                            <ipxact:subPort>
                                <ipxact:name>strb</ipxact:name>
                                <ipxact:wire>
                                    <ipxact:direction>in</ipxact:direction>
                                    <ipxact:vectors>
                                        <ipxact:vector>
                                            <ipxact:left>3</ipxact:left>
                                            <ipxact:right>0</ipxact:right>
                                        </ipxact:vector>
                                    </ipxact:vectors>
                                    <ipxact:powerConstraints>
                                        <ipxact:powerConstraint>

```

```

                <ipxact:powerDomainRef>PD1</
        ipxact:powerDomainRef>
            </ipxact:powerConstraint>
        </ipxact:powerConstraints>
    </ipxact:wire>
</ipxact:subPort>
<ipxact:subPort>
    <ipxact:name>ready</ipxact:name>
    <ipxact:wire>
        <ipxact:direction>out</ipxact:direction>
        <ipxact:powerConstraints>
            <ipxact:powerConstraint>
                <ipxact:powerDomainRef>PD1</
        ipxact:powerDomainRef>
            </ipxact:powerConstraint>
        </ipxact:powerConstraints>
    </ipxact:wire>
</ipxact:subPort>
<ipxact:subPort>
    <ipxact:name>rdata</ipxact:name>
    <ipxact:wire>
        <ipxact:direction>out</ipxact:direction>
        <ipxact:vectors>
            <ipxact:vector>
                <ipxact:left>31</ipxact:left>
                <ipxact:right>0</ipxact:right>
            </ipxact:vector>
        </ipxact:vectors>
        <ipxact:powerConstraints>
            <ipxact:powerConstraint>
                <ipxact:powerDomainRef>PD1</
        ipxact:powerDomainRef>
            </ipxact:powerConstraint>
        </ipxact:powerConstraints>
    </ipxact:wire>
</ipxact:subPort>
<ipxact:subPort>
    <ipxact:name>slverr</ipxact:name>
    <ipxact:wire>
        <ipxact:direction>out</ipxact:direction>
        <ipxact:powerConstraints>
            <ipxact:powerConstraint>
                <ipxact:powerDomainRef>PD1</
        ipxact:powerDomainRef>
            </ipxact:powerConstraint>
        </ipxact:powerConstraints>
    </ipxact:wire>
</ipxact:subPort>
</ipxact:subPorts>
<ipxact:structPortTypeDefs>
    <!-- Link: structPortTypeDef: see 6.15.23 component
structPortTypeDef -->
    <ipxact:structPortTypeDef>
        <ipxact:typeName>apb.t</ipxact:typeName>
        <ipxact:typeDefinition>apb.i.sv</
    ipxact:typeDefinition>
        <ipxact:typeParameters>
            <ipxact:typeParameter usageType="typed"
parameterId="RANGE_id" type="int">

```

```

        <ipxact:name>RANGE</ipxact:name>
        <ipxact:value>4096</ipxact:value>
        </ipxact:typeParameter>
    </ipxact:typeParameters>
    <ipxact:role>modport</ipxact:role>
    <ipxact:viewRef>RTLview</ipxact:viewRef>
    </ipxact:structPortTypeDef>
    </ipxact:structPortTypeDefs>
    </ipxact:structured>
    </ipxact:port>
</ipxact:ports>
</ipxact:model>
<!-- LINK: componentGenerators: see <a href="#">6.16, Component generators -->
<ipxact:componentGenerators>
    <ipxact:componentGenerator>
        <ipxact:name>SimpleComponentGenerator</ipxact:name>
        <ipxact:description>Simple TGI based component generator</
ipxact:description>
        <ipxact:phase>1</ipxact:phase>
        <ipxact:parameters>
            <ipxact:parameter parameterId="genParm1" prompt="Parm 1"
type="shortint" resolve="user">
                <ipxact:name>genParm1</ipxact:name>
                <ipxact:description>First generator parameter.</
ipxact:description>
                <ipxact:value>1</ipxact:value>
            </ipxact:parameter>
            <ipxact:parameter parameterId="genParm2" prompt="Parm 2"
type="shortint" resolve="user">
                <ipxact:name>genParm2</ipxact:name>
                <ipxact:description>Second generator parameter.</
ipxact:description>
                <ipxact:value>2</ipxact:value>
            </ipxact:parameter>
        </ipxact:parameters>
        <ipxact:apiType>TGI_2022_BASE</ipxact:apiType>
        <!-- By convention, generator invoked and passed parameters as
follows: -->
        <!-- generator.sh -genParm1 <value> -genParm2 <value> -->
        <ipxact:generatorExe>../bin/generator.sh</ipxact:generatorExe>
    </ipxact:componentGenerator>
</ipxact:componentGenerators>
<!-- LINK: choices: see <a href="#">C.8, Choices -->
<ipxact:choices>
    <ipxact:choice>
        <ipxact:name>bitsize</ipxact:name>
        <ipxact:enumeration text="32 bits">32</ipxact:enumeration>
        <ipxact:enumeration text="64 bits">64</ipxact:enumeration>
    </ipxact:choice>
</ipxact:choices>
<!-- LINK: fileSets: see <a href="#">6.17, File sets -->
<ipxact:fileSets>
    <!-- LINK: fileSet: see 6.17.1, File sets -->
    <ipxact:fileSet>
        <ipxact:name>VerilogFiles</ipxact:name>
        <!-- LINK: file: see 6.17.2, file -->
        <ipxact:file>
            <ipxact:name>../src/component.v</ipxact:name>
            <ipxact:fileType>verilogSource</ipxact:fileType>

```

```

        <ipxact:isStructural>true</ipxact:isStructural>
    </ipxact:file>
    <ipxact:file>
        <ipxact:name>../src/component.v</ipxact:name>
        <ipxact:fileType>verilogSource</ipxact:fileType>
    </ipxact:file>
</ipxact:fileSet>
<ipxact:fileSet>
    <ipxact:name>SystemCFiles</ipxact:name>
    <ipxact:file>
        <ipxact:name>../src/component.C</ipxact:name>
        <ipxact:fileType>systemCSource-2.2</ipxact:fileType>
    </ipxact:file>
</ipxact:fileSet>
</ipxact:fileSets>
<!-- LINK: clearboxElements: see 6.18, Clear box elements -->
<ipxact:clearboxElements>
    <ipxact:clearboxElement>
        <ipxact:name>ImportantInternalSignal</ipxact:name>
        <ipxact:description>Defines access point for forcing sim values</ipxact:description>
        <ipxact:clearboxType>signal</ipxact:clearboxType>
        <ipxact:driveable>true</ipxact:driveable>
    </ipxact:clearboxElement>
</ipxact:clearboxElements>
<!-- LINK: cpus: see 6.20, CPUs -->
<ipxact:cpus>
    <ipxact:cpu>
        <ipxact:name>cpuDefn</ipxact:name>
        <ipxact:range>4*(2**30)</ipxact:range>
        <ipxact:width>32</ipxact:width>
        <ipxact:memoryMapRef>cpuDefn_MemoryMap</ipxact:memoryMapRef>
    </ipxact:cpu>
</ipxact:cpus>
<!-- LINK: otherClockDrivers: see 6.15.18, Other clock drivers -->
<ipxact:otherClockDrivers>
    <ipxact:otherClockDriver clockName="virtualClock1">
        <ipxact:clockPeriod>10</ipxact:clockPeriod>
        <ipxact:clockPulseOffset>5</ipxact:clockPulseOffset>
        <ipxact:clockPulseValue>1</ipxact:clockPulseValue>
        <ipxact:clockPulseDuration>5</ipxact:clockPulseDuration>
    </ipxact:otherClockDriver>
</ipxact:otherClockDrivers>
<!-- LINK: resetTypes: see 6.21, Reset types -->
<ipxact:resetTypes>
    <ipxact:resetType>
        <ipxact:name>SOFT</ipxact:name>
        <ipxact:displayName>Soft Reset</ipxact:displayName>
    </ipxact:resetType>
</ipxact:resetTypes>
<!-- LINK: parameters: see C.21, parameters -->
<ipxact:parameters>
    <ipxact:parameter parameterId="comp_dual_mode" prompt="Dual Mode Supported" type="bit" resolve="user">
        <ipxact:name>comp_dual_mode</ipxact:name>
        <ipxact:description>Indicates dual mode support is desired.</ipxact:description>
        <ipxact:value>1</ipxact:value>
    </ipxact:parameter>

```

```

<!-- Parameter leveraging an enumeration 'choice' -->
<ipxact:parameter parameterId="addrBits" choiceRef="bitsize"
prompt="Address Bits" type="shortint" resolve="user">
    <ipxact:name>addrBits</ipxact:name>
    <ipxact:value>32</ipxact:value>
</ipxact:parameter>
</ipxact:parameters>
<!-- LINK: assertions: see C.5, assertions -->
<ipxact:assertions>
    <ipxact:assertion>
        <ipxact:name>ArbitraryAssertion1</ipxact:name>
        <ipxact:description>Arbitrary assertion to show syntax. Must
evaluate to true.</ipxact:description>
        <ipxact:assert>addrBits == 32 || !comp_dual_mode</ipxact:assert>
    </ipxact:assertion>
</ipxact:assertions>
<!-- LINK: vendorExtensions: see C.27, vendorExtensions -->
<ipxact:vendorExtensions>
    <!-- Simple example showing elements from a separate name space added
via vendorExtensions. -->
    <!-- The namespace must be defined (xmlns:<ns>=<URL>) at top or within
first usage. -->
    <sampleNS:extraElement xmlns:sampleNS="http://www.nosuchURL.org/
Schema">
        <sampleNS:anotherElement sampleNS:attributeName="XYZ"/>
    </sampleNS:extraElement>
</ipxact:vendorExtensions>
</ipxact:component>

```

I.7 design

```

<?xml version="1.0"?>
<!--
// Example design used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document.
-->
<!-- LINK: design: see 7.1, Design -->
<ipxact:design xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/1685-
2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
    <!-- LINK: documentNameGroup: C.11, documentName group -->
    <ipxact:vendor>accellera.org</ipxact:vendor>
    <ipxact:library>Sample</ipxact:library>
    <ipxact:name>SampleDesign</ipxact:name>
    <ipxact:version>1.0</ipxact:version>
    <!-- Three instances U1, U2, and U3 -->
    <ipxact:componentInstances>
        <!-- LINK: componentInstance: see 7.2, Design component instances -->
        <ipxact:componentInstance>
            <ipxact:instanceName>U1</ipxact:instanceName>
            <ipxact:componentRef vendor="accellera.org" library="Sample"
name="SampleComponent" version="1.0">
                <ipxact:configurableElementValues>
                    <ipxact:configurableElementValue
referenceId="comp_dual_mode">0</ipxact:configurableElementValue>

```

```

        </ipxact:configurableElementValues>
        </ipxact:componentRef>
        <ipxact:powerDomainLinks>
            <ipxact:powerDomainLink>
                <ipxact:externalPowerDomainReference>PD_top</
ipxact:externalPowerDomainReference>
                <ipxact:internalPowerDomainReference>PD1</
ipxact:internalPowerDomainReference>
            </ipxact:powerDomainLink>
            <ipxact:powerDomainLink>
                <ipxact:externalPowerDomainReference>PD_top</
ipxact:externalPowerDomainReference>
                <ipxact:internalPowerDomainReference>PD2</
ipxact:internalPowerDomainReference>
            </ipxact:powerDomainLink>
        </ipxact:componentInstance>
        <ipxact:componentInstance>
            <ipxact:instanceName>U2</ipxact:instanceName>
            <ipxact:componentRef vendor="accelera.org" library="Sample" name="SampleComponent" version="1.0"/>
            <ipxact:powerDomainLinks>
                <ipxact:powerDomainLink>
                    <ipxact:externalPowerDomainReference>PD_top</
ipxact:externalPowerDomainReference>
                    <ipxact:internalPowerDomainReference>PD1</
ipxact:internalPowerDomainReference>
                </ipxact:powerDomainLink>
                <ipxact:powerDomainLink>
                    <ipxact:externalPowerDomainReference>PD_top</
ipxact:externalPowerDomainReference>
                    <ipxact:internalPowerDomainReference>PD2</
ipxact:internalPowerDomainReference>
                </ipxact:powerDomainLink>
            </ipxact:componentInstance>
            <ipxact:componentInstance>
                <ipxact:instanceName>U3</ipxact:instanceName>
                <ipxact:componentRef vendor="accelera.org" library="Sample" name="SampleComponent" version="1.0"/>
                <ipxact:powerDomainLinks>
                    <ipxact:powerDomainLink>
                        <ipxact:externalPowerDomainReference>PD_top</
ipxact:externalPowerDomainReference>
                        <ipxact:internalPowerDomainReference>PD1</
ipxact:internalPowerDomainReference>
                    </ipxact:powerDomainLink>
                    <ipxact:powerDomainLink>
                        <ipxact:externalPowerDomainReference>PD_top</
ipxact:externalPowerDomainReference>
                        <ipxact:internalPowerDomainReference>PD2</
ipxact:internalPowerDomainReference>
                    </ipxact:powerDomainLink>
                </ipxact:componentInstance>
            </ipxact:componentInstances>
        <ipxact:interconnections>
            <!-- Simple initiator to target interface connection -->
            <!-- LINK: interconnection: see 7.3.1, interconnection -->

```

```

<ipxact:interconnection>
    <ipxact:name>TwoIntfs</ipxact:name>
    <!-- LINK: activeInterface: see 7.4, Active, hierarchical,
monitored, and monitor interfaces -->
    <ipxact:activeInterface componentInstanceRef="U1"
busRef="Initiator"/>
    <ipxact:activeInterface componentInstanceRef="U2" busRef="Target"/>
</ipxact:interconnection>
<!-- Broadcast interface from a target (one to many connection) -->
<ipxact:interconnection>
    <ipxact:name>BroadcastConnection</ipxact:name>
    <ipxact:activeInterface componentInstanceRef="U1" busRef="Target"/>
    <ipxact:hierInterface busRef="Target0"/>
    <ipxact:hierInterface busRef="Target1"/>
</ipxact:interconnection>
<!-- Export Initiator interface -->
<ipxact:interconnection>
    <ipxact:name>Export</ipxact:name>
    <ipxact:activeInterface componentInstanceRef="U3"
busRef="Initiator"/>
    <ipxact:hierInterface busRef="Initiator"/>
</ipxact:interconnection>
<!-- Export APB interface -->
<ipxact:interconnection>
    <ipxact:name>ExportAPB</ipxact:name>
    <ipxact:activeInterface componentInstanceRef="U3" busRef="APB"/>
    <ipxact:hierInterface busRef="APB"/>
</ipxact:interconnection>
<!-- LINK: monitorInterconnection: 7.3.2 monitorInterconnection -->
<!-- Monitor interface connection -->
<ipxact:monitorInterconnection>
    <ipxact:name>MonitorConnection</ipxact:name>
    <ipxact:monitoredActiveInterface componentInstanceRef="U2"
busRef="Target"/>
    <ipxact:monitorInterface componentInstanceRef="U2"
busRef="Monitor"/>
    </ipxact:monitorInterconnection>
</ipxact:interconnections>
<ipxact:adHocConnections>
    <!-- Simple two pin connection. Range not mentioned on anotherPort -->
    <!-- which implies all bits are connected -->
    <!-- LINK: adHocConnection: see 7.5, Design ad hoc connections -->
<ipxact:adHocConnection>
    <ipxact:name>SimpleWire</ipxact:name>
    <ipxact:portReferences>
        <!-- LINK: internalPortReference: see 7.6, Port references -->
        <ipxact:internalPortReference componentInstanceRef="U1"
portRef="anotherPort"/>
        <ipxact:internalPortReference componentInstanceRef="U2"
portRef="status">
            <ipxact:partSelect>
                <ipxact:range>
                    <ipxact:left>7</ipxact:left>
                    <ipxact:right>4</ipxact:right>
                </ipxact:range>
            </ipxact:partSelect>
        </ipxact:internalPortReference>
    </ipxact:portReferences>
</ipxact:adHocConnection>

```

```

<!-- Tie input bits to a constant -->
<ipxact:adHocConnection>
    <ipxact:name>TieOff</ipxact:name>
    <ipxact:tiedValue>3'h2</ipxact:tiedValue>
    <ipxact:portReferences>
        <ipxact:internalPortReference componentInstanceRef="U2"
portRef="anotherPort">
            <ipxact:partSelect>
                <ipxact:range>
                    <ipxact:left>2</ipxact:left>
                    <ipxact:right>0</ipxact:right>
                </ipxact:range>
            </ipxact:partSelect>
        </ipxact:internalPortReference>
    </ipxact:portReferences>
</ipxact:adHocConnection>
<!-- Tie output bits to open -->
<ipxact:adHocConnection>
    <ipxact:name>TieOpen</ipxact:name>
    <ipxact:tiedValue>open</ipxact:tiedValue>
    <ipxact:portReferences>
        <ipxact:internalPortReference componentInstanceRef="U1"
portRef="status"/>
    </ipxact:portReferences>
</ipxact:adHocConnection>
</ipxact:adHocConnections>
</ipxact:design>

```

I.8 designConfiguration

```

<?xml version="1.0"?>
<!--
// Example designConfiguration used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document.
-->
<!-- LINK: designConfiguration: see 11.2, designConfiguration -->
<ipxact:designConfiguration xmlns:ipxact="http://www.accellera.org/XMLSchema/
IPXACT/1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
<!-- LINK: documentNameGroup: C.11, documentName group -->
<ipxact:vendor>accellera.org</ipxact:vendor>
<ipxact:library>Sample</ipxact:library>
<ipxact:name>SampleDesignConfiguration</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<ipxact:generatorChainConfiguration vendor="accellera.org"
library="Sample" name="SampleGeneratorChain" version="1.0">
    <ipxact:configurableElementValues>
        <ipxact:configurableElementValue referenceId="genParm1">7</
ipxact:configurableElementValue>
    </ipxact:configurableElementValues>
</ipxact:generatorChainConfiguration>
<!-- LINK: interconnectionConfiguration: see 11.3,
interconnectionConfiguration -->
<ipxact:interconnectionConfiguration>
    <ipxact:interconnectionRef>Export</ipxact:interconnectionRef>

```

```

<ipxact:abstractorInstances>
    <!-- LINK: abstractorInstance: see 11.4, abstractor instance -->
    <ipxact:abstractorInstance>
        <ipxact:instanceName>U_tlm2rtl</ipxact:instanceName>
        <ipxact:abstractorRef vendor="accellera.org" library="Sample"
name="SampleAbstractor" version="1.0"/>
        <ipxact:viewName>default</ipxact:viewName>
    </ipxact:abstractorInstance>
</ipxact:abstractorInstances>
</ipxact:interconnectionConfiguration>
<!-- Select RTL for U1, U2, and TLM for U3 -->
<!-- LINK: viewConfiguration: see 11.5, viewConfiguration -->
<ipxact:viewConfiguration>
    <ipxact:instanceName>U1</ipxact:instanceName>
    <ipxact:view viewRef="RTLview"/>
</ipxact:viewConfiguration>
<ipxact:viewConfiguration>
    <ipxact:instanceName>U2</ipxact:instanceName>
    <ipxact:view viewRef="RTLview"/>
</ipxact:viewConfiguration>
<ipxact:viewConfiguration>
    <ipxact:instanceName>U3</ipxact:instanceName>
    <ipxact:view viewRef="TLMview">
        <ipxact:configurableElementValues>
            <ipxact:configurableElementValue referenceId="vdp">1</
ipxact:configurableElementValue>
        </ipxact:configurableElementValues>
    </ipxact:view>
</ipxact:viewConfiguration>
</ipxact:designConfiguration>

```

I.9 fieldAccessPolicyDefinitions

```

<?xml version="1.0"?>
<!--
// Example typeDefinitions used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document. These definitions are field access policy
// definitions.
-->
<!-- LINK: typeDefinitions: see 9.1, Type definitions -->
<ipxact:typeDefinitions xmlns:ipxact="http://www.accellera.org/XMLSchema/
IPXACT/1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
    <!-- LINK: documentNameGroup: C.11, documentName group -->
    <ipxact:vendor>accellera.org</ipxact:vendor>
    <ipxact:library>Sample</ipxact:library>
    <ipxact:name>FieldAccessPolicyDefinitions</ipxact:name>
    <ipxact:version>1.0</ipxact:version>
    <ipxact:displayName>Register Field Access Definitions</ipxact:displayName>
    <ipxact:shortDescription>Example register field access definitions</
ipxact:shortDescription>
    <ipxact:description>Example field access policy definitions used in the IP-
XACT standard.</ipxact:description>
    <ipxact:fieldAccessPolicyDefinitions>

```

```

<!-- LINK: fieldAccessPolicyDefinition: 9.2, Field access policy
definition -->
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>NOACCESS</ipxact:name>
  <ipxact:access>no-access</ipxact:access>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>RO</ipxact:name>
  <ipxact:access>read-only</ipxact:access>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>RC</ipxact:name>
  <ipxact:access>read-only</ipxact:access>
  <ipxact:readAction>clear</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>RS</ipxact:name>
  <ipxact:access>read-only</ipxact:access>
  <ipxact:readAction>set</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>WO</ipxact:name>
  <ipxact:access>write-only</ipxact:access>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>WOC</ipxact:name>
  <ipxact:access>write-only</ipxact:access>
  <ipxact:modifiedWriteValue>clear</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>WOS</ipxact:name>
  <ipxact:access>write-only</ipxact:access>
  <ipxact:modifiedWriteValue>set</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>W01</ipxact:name>
  <ipxact:access>writeOnce</ipxact:access>
  <ipxact:modifiedWriteValue>set</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>RW</ipxact:name>
  <ipxact:access>read-write</ipxact:access>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>W1C</ipxact:name>
  <ipxact:access>read-write</ipxact:access>
  <ipxact:modifiedWriteValue>oneToClear</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>W1S</ipxact:name>
  <ipxact:access>read-write</ipxact:access>
  <ipxact:modifiedWriteValue>oneToSet</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>W1T</ipxact:name>
  <ipxact:access>read-write</ipxact:access>
  <ipxact:modifiedWriteValue>oneToToggle</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>

```

```

<ipxact:name>W0C</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>zeroToClear</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>W1C</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>oneToClear</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>W1T</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>oneToToggle</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>WC</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>clear</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>WS</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>set</ipxact:modifiedWriteValue>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>WRC</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:readAction>clear</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>W1SRC</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>oneToSet</ipxact:modifiedWriteValue>
<ipxact:readAction>clear</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>W0SRC</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>zeroToSet</ipxact:modifiedWriteValue>
<ipxact:readAction>clear</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>WSRC</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>set</ipxact:modifiedWriteValue>
<ipxact:readAction>clear</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>WRS</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:readAction>set</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
<ipxact:name>W1CRS</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>oneToClear</ipxact:modifiedWriteValue>
<ipxact:readAction>set</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>

```

```

<ipxact:name>W0CRS</ipxact:name>
<ipxact:access>read-write</ipxact:access>
<ipxact:modifiedWriteValue>zeroToClear</ipxact:modifiedWriteValue>
<ipxact:readAction>set</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>WCRS</ipxact:name>
  <ipxact:access>read-write</ipxact:access>
  <ipxact:modifiedWriteValue>clear</ipxact:modifiedWriteValue>
  <ipxact:readAction>set</ipxact:readAction>
</ipxact:fieldAccessPolicyDefinition>
<ipxact:fieldAccessPolicyDefinition>
  <ipxact:name>W1</ipxact:name>
  <ipxact:access>read-writeOnce</ipxact:access>
</ipxact:fieldAccessPolicyDefinition>
</ipxact:fieldAccessPolicyDefinitions>
</ipxact:typeDefinitions>

```

I.10 generatorChain

```

<?xml version="1.0"?>
<!--
// Example generatorChain used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document
-->
<!-- LINK: generatorChain: see 10.1, generatorChain -->
<ipxact:generatorChain xmlns:ipxact="http://www.accelera.org/XMLSchema/
IPXACT/1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accelera.org/XMLSchema/IPXACT/1685-2022
http://www.accelera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
<ipxact:vendor>accelera.org</ipxact:vendor>
<ipxact:library>Sample</ipxact:library>
<ipxact:name>SampleGeneratorChain</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<ipxact:displayName>Sample Generator Chain</ipxact:displayName>
<ipxact:description>Example generator chain used in the IP-XACT standard.</
ipxact:description>
<!-- Include generators with the indicated group names -->
<!-- LINK: generatorChainSelector: see 10.2, generatorChainSelector -->
<ipxact:generatorChainSelector>
  <ipxact:groupSelector multipleGroupSelectionOperator="and">
    <ipxact:name>HDLAnalysis</ipxact:name>
    <ipxact:name>HDLElaboration</ipxact:name>
  </ipxact:groupSelector>
</ipxact:generatorChainSelector>
<!-- Include generator with the indicated VLVN -->
<ipxact:generatorChainSelector>
  <ipxact:generatorChainRef vendor="accelera.org" library="Sample"
name="SampleGeneratorChainForReference" version="1.0"/>
</ipxact:generatorChainSelector>
<!-- Include component generator with the indicated group name -->
<!-- LINK: componentGeneratorSelector: see 10.3, generatorChain component
selector -->
<ipxact:componentGeneratorSelector>
  <ipxact:groupSelector>
    <ipxact:name>DocGeneration</ipxact:name>

```

```

        </ipxact:groupSelector>
        </ipxact:componentGeneratorSelector>
        <!-- Define a generator explicitly within the generator chain -->
        <!-- LINK: generator: see 10.4, generatorChain generator -->
        <ipxact:generator hidden="false">
            <ipxact:name>myGenerator</ipxact:name>
            <ipxact:displayName>My Generator</ipxact:displayName>
            <ipxact:description>This is my generator</ipxact:description>
            <ipxact:phase>100.0</ipxact:phase>
            <ipxact:parameters>
                <ipxact:parameter parameterId="genParm1" prompt="Parm 1"
type="shortint" resolve="user">
                    <ipxact:name>genParm1</ipxact:name>
                    <ipxact:description>First generator parameter.</
ipxact:description>
                    <ipxact:value>1</ipxact:value>
                </ipxact:parameter>
            </ipxact:parameters>
            <ipxact:apiType>TGI_2022_BASE</ipxact:apiType>
            <ipxact:apiService>SOAP</ipxact:apiService>
            <ipxact:transportMethods>
                <ipxact:transportMethod>file</ipxact:transportMethod>
            </ipxact:transportMethods>
            <ipxact:generatorExe>../bin/run.sh</ipxact:generatorExe>
        </ipxact:generator>
        <ipxact:chainGroup>implementation</ipxact:chainGroup>
    </ipxact:generatorChain>

```

I.11 typeDefinitions

```

<?xml version="1.0"?>
<!--
// Example typeDefinitions used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to section numbers in
// the standard definition document.
-->
<!-- LINK: typeDefinitions: see 9.1, Type definitions -->
<ipxact:typeDefinitions xmlns:ipxact="http://www.accellera.org/XMLSchema/
IPXACT/1685-2022" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/1685-2022
http://www.accellera.org/XMLSchema/IPXACT/1685-2022/index.xsd">
    <!-- LINK: documentNameGroup: C.11, documentName group -->
    <ipxact:vendor>accellera.org</ipxact:vendor>
    <ipxact:library>Sample</ipxact:library>
    <ipxact:name>SampleTypeDefinitions</ipxact:name>
    <ipxact:version>1.0</ipxact:version>
    <ipxact:displayName>Sample Type Definitions</ipxact:displayName>
    <ipxact:shortDescription>Example typeDefinitions</
ipxact:shortDescription>
    <ipxact:description>Example typeDefinitions used in the IP-XACT
standard.</ipxact:description>
    <ipxact:externalTypeDefinitions>
        <ipxact:name>fieldAccessTypes</ipxact:name>
        <ipxact:typeDefinitionsRef vendor="accellera.org" library="Sample"
name="FieldAccessPolicyDefinitions" version="1.0"/>
    </ipxact:externalTypeDefinitions>
    <ipxact:modes>

```

```

<ipxact:mode>
  <ipxact:name>Normal</ipxact:name>
</ipxact:mode>
</ipxact:modes>
<ipxact:views>
  <ipxact:view>
    <ipxact:name>RTLview</ipxact:name>
  </ipxact:view>
</ipxact:views>
<ipxact:enumerationDefinitions>
  <!-- LINK: enumerationDefinition: see 9.3, Enumeration definition -->
  <ipxact:enumerationDefinition>
    <ipxact:name>setpos</ipxact:name>
    <ipxact:width>4</ipxact:width>
    <ipxact:enumeratedValue>
      <ipxact:name>SetPos0</ipxact:name>
      <ipxact:value>'h1</ipxact:value>
    </ipxact:enumeratedValue>
    <ipxact:enumeratedValue>
      <ipxact:name>SetPos1</ipxact:name>
      <ipxact:value>'h2</ipxact:value>
    </ipxact:enumeratedValue>
  </ipxact:enumerationDefinition>
</ipxact:enumerationDefinitions>
<ipxact:fieldDefinitions>
  <!-- LINK: fieldDefinition: see 9.4, Field definition -->
  <!-- fieldDefinition with reference to fieldAccessPolicyDefinition -->
  <ipxact:fieldDefinition>
    <ipxact:name>status</ipxact:name>
    <ipxact:bitWidth>4</ipxact:bitWidth>
    <ipxact:resets>
      <ipxact:reset resetTypeRef="SOFT">
        <ipxact:value>'h0</ipxact:value>
        <ipxact:mask>'hf</ipxact:mask>
      </ipxact:reset>
    </ipxact:resets>
    <ipxact:fieldAccessPolicies>
      <ipxact:fieldAccessPolicy>
        <!-- Reference to fieldAccessPolicyDefinition to describe
        this fieldAccessPolicy -->
        <ipxact:fieldAccessPolicyDefinitionRef
        typeDefinitions="fieldAccessTypes">RC</
        ipxact:fieldAccessPolicyDefinitionRef>
      </ipxact:fieldAccessPolicy>
    </ipxact:fieldAccessPolicies>
  </ipxact:fieldDefinition>
  <!-- fieldDefinition with inline fieldAccessPolicy description -->
  <ipxact:fieldDefinition>
    <ipxact:name>clr</ipxact:name>
    <ipxact:bitWidth>1</ipxact:bitWidth>
    <ipxact:fieldAccessPolicies>
      <ipxact:fieldAccessPolicy>
        <!-- Inline description for this fieldAccessPolicy -->
        <ipxact:access>read-write</ipxact:access>
        <ipxact:modifiedWriteValue>oneToClear</
        ipxact:modifiedWriteValue>
      </ipxact:fieldAccessPolicy>
    </ipxact:fieldAccessPolicies>
  </ipxact:fieldDefinition>

```

```

</ipxact:fieldDefinitions>
<ipxact:registerDefinitions>
    <!-- LINK: registerDefinition: see 9.5, Register definition -->
    <ipxact:registerDefinition>
        <ipxact:name>reg</ipxact:name>
        <ipxact:size>32</ipxact:size>
        <ipxact:accessPolicies>
            <ipxact:accessPolicy>
                <ipxact:modeRef priority="1">Normal</ipxact:modeRef>
                <ipxact:access>read-write</ipxact:access>
            </ipxact:accessPolicy>
        </ipxact:accessPolicies>
        <ipxact:field>
            <ipxact:name>fld</ipxact:name>
            <ipxact:accessHandles>
                <ipxact:accessHandle>
                    <ipxact:viewRef>RTLview</ipxact:viewRef>
                    <ipxact:slices>
                        <ipxact:slice>
                            <ipxact:pathSegments>
                                <!-- Path segment using $sformatf() and
$ipxact_index_value() functions -->
                                <!-- to avoid unrolling of multi-dimensional
array to describe HDL path -->
                                <ipxact:pathSegment>$sformatf("FIELD_%d",
$ipxact_index_value(i) * $ipxact_index_value(j))</ipxact:pathSegment>
                            </ipxact:pathSegments>
                        </ipxact:slice>
                    </ipxact:slices>
                </ipxact:accessHandle>
            </ipxact:accessHandles>
            <ipxact:array>
                <ipxact:dim indexVar="i">4</ipxact:dim>
                <ipxact:dim indexVar="j">3</ipxact:dim>
            </ipxact:array>
            <ipxact:bitOffset>0</ipxact:bitOffset>
            <ipxact:bitWidth>32</ipxact:bitWidth>
        </ipxact:field>
    </ipxact:registerDefinition>
</ipxact:registerDefinitions>
<ipxact:registerFileDefinitions>
    <!-- LINK: registerFileDefinition: see 9.6, Register file definition -->
    <ipxact:registerFileDefinition>
        <ipxact:name>regfile</ipxact:name>
        <!-- range is expressed in addressUnitBits which is not defined -->
        <ipxact:range>'h8</ipxact:range>
        <ipxact:register>
            <ipxact:name>reg0</ipxact:name>
            <!-- addressOffset is expressed in addressUnitBits of the
registerFileDefinition -->
            <ipxact:addressOffset>'h0</ipxact:addressOffset>
            <ipxact:size>32</ipxact:size>
            <ipxact:field>
                <ipxact:name>fld</ipxact:name>
                <ipxact:bitOffset>'h0</ipxact:bitOffset>
                <ipxact:bitWidth>32</ipxact:bitWidth>
            </ipxact:field>
        </ipxact:register>
    </ipxact:registerFileDefinition>

```

```

<ipxact:name>reg1</ipxact:name>
<ipxact:addressOffset>'h4</ipxact:addressOffset>
<ipxact:size>32</ipxact:size>
<ipxact:field>
    <ipxact:name>fld</ipxact:name>
    <ipxact:bitOffset>'h0</ipxact:bitOffset>
    <ipxact:bitWidth>32</ipxact:bitWidth>
</ipxact:field>
</ipxact:register>
<ipxact:addressUnitBits>8</ipxact:addressUnitBits>
</ipxact:registerFileDefinition>
</ipxact:registerFileDefinitions>
<ipxact:addressBlockDefinitions>
    <!-- LINK: addressBlockDefinition: see 9.7, Address block definition -->
    <ipxact:addressBlockDefinition>
        <ipxact:name>block</ipxact:name>
        <!-- range is expressed in addressUnitBits of the
addressBlockDefinition -->
        <ipxact:range>'h1000</ipxact:range>
        <ipxact:width>32</ipxact:width>
        <ipxact:addressUnitBits>8</ipxact:addressUnitBits>
    </ipxact:addressBlockDefinition>
</ipxact:addressBlockDefinitions>
<ipxact:bankDefinitions>
    <!-- LINK: bankDefinition: see 9.8, Bank definition -->
    <ipxact:bankDefinition>
        <ipxact:name>bank</ipxact:name>
        <ipxact:addressBlock>
            <ipxact:name>banked_block</ipxact:name>
            <!-- range is expressed in addressUnitBits of the bankDefinition
-->
            <ipxact:range>'h1000</ipxact:range>
            <ipxact:width>32</ipxact:width>
        </ipxact:addressBlock>
        <ipxact:bank bankAlignment="parallel">
            <ipxact:name>banked_bank</ipxact:name>
            <ipxact:addressBlock>
                <ipxact:name>banked_bank_block</ipxact:name>
                <ipxact:range>'h1000</ipxact:range>
                <ipxact:width>32</ipxact:width>
            </ipxact:addressBlock>
        </ipxact:bank>
        <ipxact:addressUnitBits>8</ipxact:addressUnitBits>
    </ipxact:bankDefinition>
</ipxact:bankDefinitions>
<ipxact:memoryMapDefinitions>
    <!-- LINK: memoryMapDefinition: see 9.9, Memory map definition -->
    <ipxact:memoryMapDefinition>
        <ipxact:name>memorymap</ipxact:name>
        <ipxact:addressBlock>
            <ipxact:name>memorymap_block</ipxact:name>
            <!-- baseAddress is expressed in addressUnitBits of the
memoryMapDefinition -->
            <ipxact:baseAddress>'h0</ipxact:baseAddress>
            <!-- range is expressed in addressUnitBits of the
memoryMapDefinition -->
            <ipxact:range>'h100</ipxact:range>
            <ipxact:width>32</ipxact:width>
        </ipxact:addressBlock>

```

```
<ipxact:addressUnitBits>8</ipxact:addressUnitBits>
</ipxact:memoryMapDefinition>
</ipxact:memoryMapDefinitions>
<ipxact:memoryRemapDefinitions>
    <!-- LINK: memoryRemapDefinition: see 9.10, Memory remap definition -->
    <ipxact:memoryRemapDefinition>
        <ipxact:name>memoryremap</ipxact:name>
        <ipxact:addressBlock>
            <ipxact:name>memoryremap_block</ipxact:name>
            <!-- baseAddress is expressed in addressUnitBits of the
            memoryRemapDefinition -->
            <ipxact:baseAddress>'h100</ipxact:baseAddress>
            <!-- range is expressed in addressUnitBits of the
            memoryRemapDefinition -->
            <ipxact:range>'h100</ipxact:range>
            <ipxact:width>32</ipxact:width>
        </ipxact:addressBlock>
    </ipxact:memoryRemapDefinition>
</ipxact:memoryRemapDefinitions>
<ipxact:resetTypes>
    <ipxact:resetType>
        <ipxact:name>SOFT</ipxact:name>
    </ipxact:resetType>
</ipxact:resetTypes>
</ipxact:typeDefinitions>
```

RAISING THE WORLD'S STANDARDS

Connect with us on:

-  **Twitter:** twitter.com/ieeesa
-  **Facebook:** facebook.com/ieeesa
-  **LinkedIn:** linkedin.com/groups/1791118
-  **Beyond Standards blog:** beyondstandards.ieee.org
-  **YouTube:** youtube.com/ieeesa

standards.ieee.org
Phone: +1 732 981 0060