INTERNATIONAL STANDARD

ISO/IEC/IEEE 31320-2

First edition
2012-09-15

Information technology — Modeling Languages —

Part 2: Syntax and Semantics for IDEF1X$_{97}$ (IDEF$_{object}$)

Technologies de l'information — Langages de modélisation — Partie 2: Syntaxe et sémantique pour IDEF1X$_{97}$ (IDEF$_{object}$)

Reference number
ISO/IEC/IEEE 31320-2:2012(E)

© IEEE 1999

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

COPYRIGHT PROTECTED DOCUMENT

© IEEE 1999

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from ISO, IEC or IEEE at the respective address below.

ISO copyright office
Case postale 56
CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
E-mail inmail@iec.ch
Web www.iec.ch

Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York
NY 10016-5997, USA
E-mail stds.ipr@ieee.org
Web www.ieee.org

Published in Switzerland

© IEEE 1999 – All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: ISO/IEC/IEEE 31320-2-2012

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

The main task of ISO/IEC JTC 1 is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is called to the possibility that implementation of this standard may require the use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. ISO/IEEE is not responsible for identifying essential patents or patent claims for which a license may be required, for conducting inquiries into the legal validity or scope of patents or patent claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance or a Patent Statement and Licensing Declaration Form, 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 ISO or the IEEE Standards Association.

ISO/IEC/IEEE 31320-2 was prepared by the Software & Systems Engineering Standards Committee of the IEEE Computer Society (as IEEE 1320.2-1998). It was adopted by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*, in parallel with its approval by the ISO/IEC national bodies, under the “fast-track procedure” defined in the Partner Standards Development Organization cooperation agreement between ISO and IEEE. IEEE is responsible for the maintenance of this document with participation and input from ISO/IEC national bodies.

ISO/IEC/IEEE 31320 consists of the following parts:

— ISO/IEC/IEEE 31320-1, *Information technology — Modeling Languages — Part 1: Syntax and Semantics for IDEF0*

— ISO/IEC/IEEE 31320-2, *Information technology — Modeling Languages — Part 2: Syntax and Semantics for IDEF1X$_{97}$ (IDEF$_{\text{object}}$)*

© IEEE 1999 – All rights reserved

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Documents
Document Sharing Hub

Full document details: ISO/IEC/IEEE 31320-2-2012

(blank page)

© IEEE 1999 – All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

IEEE Std 1320.2-1998(R2004)

IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEF1X$_{97}$ (*IDEF*$_{object}$)

Sponsor
Software Engineering Standards Committee
of the
IEEE Computer Society

Reaffirmed 25 March 2004
Approved Approved 25 June 1998
IEEE-SA Standards Board

Abstract: IDEF1X$_{97}$ consists of two conceptual modeling languages. The key-style language supports data/information modeling and is downward compatible with the US government's 1993 standard, FIPS PUB 184. The identity-style language is based on the object model with declarative rules and constraints. IDEF1X$_{97}$ identity style includes constructs for the distinct but related components of object abstraction: interface, requests, and realization; utilizes graphics to state the interface; and defines a declarative, directly executable Rule and Constraint Language for requests and realizations. IDEF1X$_{97}$ conceptual modeling supports implementation by relational databases, extended relational databases, object databases, and object programming languages. IDEF1X$_{97}$ is formally defined in terms of first order logic. A procedure is given whereby any valid IDEF1X$_{97}$ model can be transformed into an equivalent theory in first order logic. That procedure is then applied to a meta model of IDEF1X$_{97}$ to define the valid set of IDEF1X$_{97}$ models.
Keywords: conceptual schema, data model, IDEF1X, IDEF1X$_{97}$, identity style, information model, key style, object model

---

The Institute of Electrical and Electronics Engineers, Inc.

345 East 47th Street, New York, NY 10017-2394, USA

Copyright ©1997 by the Institute of Electrical and Electronics Engineers, Inc.

All rights reserved. Published 1997. Printed in the United States of America

Print ISBN 0-7381-0341-1 SH94663

PDF ISBN 0-7381-1405-7 SS94663

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.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

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. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, 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.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: 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 with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Full document details: ISO/IEC/IEEE 31320-2-2012

Introduction

[This introduction is not a part of IEEE Std 1320.2-1998, IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEF1X$_{97}$ *(IDEF$_{object}$)*.]

Background

The need for semantic models to represent conceptual schemas was recognized by the US Air Force in the mid 1970s as a result of the Integrated Computer Aided Manufacturing (ICAM) Program. The objective of this program was to increase manufacturing productivity through the systematic application of computer technology. The ICAM program identified a need for better analysis and communication techniques for people involved in improving manufacturing productivity. As a result, the ICAM program developed a series of techniques known as the ICAM Definition (IDEF) methods, which included the following:

a) IDEF0, a technique used to produce a "function model," which is a structured representation of the activities or processes within the environment or system.
b) IDEF1, a technique used to produce an "information model," which represents the structure and semantics of information within the environment or system.
c) IDEF2, a technique used to produce a "dynamics model," which represents the time-varying behavioral characteristics of the environment or system.

IDEF0 and IDEF1X (the successor to IDEF1) continue to be used extensively in various government and industry settings. IDEF2 is no longer used to any significant extent.

The initial approach to IDEF information modeling (IDEF1) was published by the ICAM program in 1981, based on current research and industry needs [B23].[1] The theoretical roots for this approach stemmed from the early work of Dr. E. F. Codd on relational theory and Dr. P. P. S. Chen on the entity-relationship model. The initial IDEF1 technique was based on the work of Dr. R. R. Brown and Mr. T. L. Ramey of Hughes Aircraft and Mr. D. S. Coleman of D. Appleton Company, with critical review and influence by Mr. C. W. Bachman, Dr. P. P. S. Chen, Dr. M. A. Melkanoff, and Dr. G. M. Nijssen.

In 1983, the US Air Force initiated the Integrated Information Support System (I^2S^2) project under the ICAM program. The objective of this project was to provide the enabling technology to integrate a network of heterogeneous computer hardware and software both logically and physically. As a result of this project and industry experience, the need for an enhanced technique for information modeling was recognized.

Application within industry had led to the development in 1982 of a Logical Database Design Technique (LDDT) by R. G. Brown of the Database Design Group. The technique was also based on the relational model of Dr. E. F. Codd and the entity-relationship model of Dr. P. P. S. Chen, with the addition of the generalization concepts of J. M. Smith and D. C. P. Smith. LDDT provided multiple levels of models and a set of graphics for representing the conceptual view of information within an enterprise. It had a high degree of overlap with IDEF1 features, included additional semantic and graphical constructs, and addressed information modeling enhancement requirements that had been identified under the I^2S^2 program. Under the technical leadership of Dr. M. E. S. Loomis of D. Appleton Company, a substantial subset of LDDT was combined with the methodology of IDEF1 and published by the ICAM program in 1985 [B15]. This technique was called IDEF1 Extended or, simply, IDEF1X.

In December 1993, the US government released a Federal Information Processing Standard (FIPS) for IDEF1X. FIPS PUB 184 [B13] was based on the ICAM program description of IDEF1X and additional features originally included in LDDT. The FIPS clarified and corrected points in the ICAM publication, sepa-

[1]The numbers in brackets correspond to those of the bibliography items listed in Annex A.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

rated language syntax and semantics definition from practice and use issues, and provided a formal first-order language definition of IDEF1X.

IEEE Std 1320.2-1998 continues the evolution of the IDEF1X language. It is driven by two needs. First, development of a national standard for the language makes the definition more accessible to organizations that do not follow US government standards and allows consideration and inclusion of features needed outside the US federal government sector. Second, the needs of the users of a standard change over time as system development techniques and available technology continue to evolve. Some users adopt new concepts earlier than others. To be valuable to the widest set of users, this standard needs to support a range of practices, from those supported by the FIPS to those that are emerging as future drivers of integration.

The change in the drivers of integration is being recognized by both government and private sector organizations. Integration involves not only data but the operations performed on that data. The emerging object modeling approaches seek to treat all activities as performed by collaborating objects that encompass both the data and the operations that can be performed against that data. There is increasing interest in these approaches in both the government and private sectors. Original work done for the National Institute of Standards and Technology (NIST) in 1994 and early 1995 by Robert G. Brown of the Database Design Group (DBDG) provides the basic elements required for a graceful evolution of IDEF1X toward full coverage of object modeling [B5].

The DBDG work analyzed the 1993 definition of IDEF1X and compared to it to the emerging consensus object model. The analysis showed that

— The concepts of the current IDEF1X were a subset of those of the object model,
— The current IDEF1X contained restrictions that are unnecessary in the object model, and
— The object model contains significant new concepts.

The work also showed that if the concepts of IDEF1X were more fully developed, the restrictions dropped, and the new concepts added, the result would be an upwardly compatible object modeling technique. The evolutionary features of IDEF1X described in this standard draw heavily from the DBDG work done for NIST.

Base documents

The following documents served as base documents for the parts of IEEE Std 1320.2-1998 indicated:

a) *From IDEF1X to IDEF$_{object}$*, 1995, by Robert G. Brown, The Database Design Group, Newport Beach, CA, is the base document for the Class and Responsibility clauses. Partial financial support was provided by the National Institute of Standards and Technology (NIST) [B5].
b) *IDEF1X$_{97}$ Rule and Constraint Language (RCL),* 1997, by Robert G. Brown, The Database Design Group, Newport Beach, CA, is the base document for the RCL clause. Partial financial support was provided by the Defense Information Systems Agency (DISA) [B6].
c) *IDEF1X$_{97}$ Formalization,* 1998, by Valdis Berzins, Naval Postgraduate School, Monterey, CA, and Robert G. Brown, The Database Design Group, Newport Beach, CA, is the base document for the Formalization clause. Partial financial support was provided by DISA and the Defense Modeling and Simulations Office (DMSO) [B7].

The IDEF1X approach

A principal objective of IDEF1X is to support integration. The “IDEF1X approach” to integration focuses on the capture, management, and use of a single semantic definition of the data resource referred to as a *conceptual schema*. The conceptual schema provides a single integrated definition of the concepts relevant to an enterprise, unbiased toward any particular application. The primary objective of this conceptual schema is to

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

provide a consistent definition of the meanings and interrelationship of concepts. This definition can then be used to integrate, share, and manage the integrity of the concepts. A conceptual schema must have three important characteristics:

— It must be consistent with the infrastructure of the business and be true across all application areas.
— It must be extendible, such that, new concepts can be defined without disruption to previously defined concepts.
— It must be transformable to both the required user views and to a variety of implementation environments.

IDEF1X is the semantic modeling technique described by IEEE Std 1320.2-1998. The IDEF1X technique was developed to meet the following requirements:

— Support the development of conceptual schemas.
— Be a coherent language.
— Be teachable.
— Be well-tested and proven.
— Be automatable.

Organization of this document

This document begins with an explanation of the scope and purpose of this version of the IDEF1X standard. Clause 1 also describes the evolution of the IDEF1X standard. It provides a context for understanding the approach and constructs presented in the rest of this standard.

Clause 2 identifies additional references that must be on hand and available to the reader of this standard for its implementation. Other documentation and related references that might be of interest to the reader or that were used in preparing this standard are included in the bibliography (see Annex A).

This document uses words in accordance with their definitions in the *Merriam-Webster's Collegiate Dictionary* [B26]. A definitions clause (see Clause 3) is provided for the convenience of those not already familiar with the terminology in question. It also contains any terminology that has specialized meaning in the context of this standard.

Clauses 4 through 6 along with 8 discuss the meaning (semantics) of each model construct that may be used within an IDEF1X model, as well as how they shall be put together to form a valid model (the syntax). Clause 7 provides a full description of the Rule and Constraint Language (RCL) specification language for an IDEF1X model.

Clause 4 introduces the language constructs of IDEF1X. The basic constructs of an IDEF1X model are

a) Things whose knowledge or behavior is relevant to the enterprise, represented by boxes;
b) Relationships between those things, represented by lines connecting the boxes;
c) Responsibilities of those things, stated as
    1) Knowledge and behavior properties, represented by names within the boxes,
    2) Realization of those responsibilities, expressed as sentences in a declarative language, and
    3) Rules, represented as constraints over property values.

These constructs are then described in detail in Clauses 5 and 6. Clause 8 discusses how the various constructs may be put together to form a model.

Two styles of IDEF1X modeling are described in this standard. Clauses 5 through 8 present the *identity style*, which extends the conceptual schema representation capabilities of IDEF1X. Identity-style

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

models describe the structural dimension of an object model and specify the collaborations among the objects. Identity-style models can be used in conjunction with dynamic modeling techniques such as those based on finite state machines.

Clause 9 describes the *key style*, which is backward-compatible with FIPS PUB 184 [B13]. This style may continue to be used to produce models that represent the structure and semantics of data within an enterprise, i.e., data (information) models.

In the process of producing FIPS PUB 184 [B13], the various graphical constructs of the IDEF1X language were formalized. In essence, these constructs had no more meaning than they had before, but they became more explicitly grounded than they had been. The formalization served to make obvious the fact that the graphical aspect of IDEF1X was not the language *per se* but only one external manifestation of it. Clause 10 presents the formalization of the IDEF1X language, revised to include the language features defined in IEEE Std 1320.2-1998. The formalization also provides a metamodel of IDEF1X. In addition to the metamodel diagram, the metamodel value classes and constraints are given. The reader may wish to use this model along with Annex D, which documents the built-in classes of the IDEF1X metamodel.

Additional normative and informative annexes provide convenient reference to supporting material:

— Annex A is a bibliography of relevant reference material.
— Annex B summarizes the differences and similarities between the version of IDEF1X documented in FIPS PUB 184 [B13] and this standard. The reader familiar with FIPS PUB 184 may wish to review this information before proceeding into the body of IEEE Std 1320.2-1998.
— Annex C presents a set of examples illustrating various aspects of identity-style modeling. These examples include the representation of two patterns from *Design* Patterns [B14], a business example that applies these patterns, some value class examples, and the translation of the TcCo model from FIPS PUB 184 [B13] into an initial identity-style model.
— Annex D documents the built-in classes of the IDEF1X language.

Throughout this standard, IEEE conventions for certain words are used:

— "Shall" means "required." For example, point 5.1.2.1 a) says "A class shall be represented as a rectangle of the shape appropriate to the class." This statement is interpreted as a mandatory requirement that a rectangle be the only acceptable way to represent a class.
— "Should" means "recommended." For example, point 8.1.3.7 a) says "If the objective of the view is that it be internally consistent, it should be possible to demonstrate that a consistent set of instances exists." This statement means that the presentation of a set of instances is highly recommended but not required for conformance.
— "May" means "permitted." For example, point 5.2.3.6 c) says that "In a sample instance table, the instance identity label may be shown to the left of the row representing the instance."
— "Can" means "is able to." For example, 7.5.3 states that "The uniqueness conditions guarantee that a message can be resolved to at most one class responsibility."

Reading the document

The $IDEF1X_{97}$ ($IDEF_{object}$) standard was developed to extend the practice of information modeling ($IDEF1X_{93}$) to object modeling. The readers of this standard can be broadly classified into at least at two distinct groups: management and technical. For each group, a different approach to the reading of this standard is recommended.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

Management readers

This standard is written on a fairly technical plane. Managers may wish to focus on the concepts that will help them manage projects that employ this new standard. For example, modeling now will include operations on enterprise knowledge as well as rules that govern state changes. For this group of readers, Clause 1 should be read first as the key to the document. Clause 1 delimits the scope and defines the purpose of the document, providing a succinct discussion of the evolution of IDEF1X and pointing out the capabilities added in $IDEF1X_{97}$. As this clause points out, $IDEF1X_{97}$ is considered a transition language that preserves the information modeling investment, provides opportunities to simplify the data/process approach, and positions the organization to move forward.

Clause 4 should be read next. This clause provides a high level summary of the language concepts, constructs, and notation. The notation is not terribly significant to management; however, many of the concepts and constructs summarized in Clause 4 will lead a management reader to further discussions of concepts and constructs found in Clauses 5 through 8.

In the past at least two separate requirement specification languages had to be used (e.g. IDEF0, "Function Modeling" and $IDEF1X_{93}$, "Information Modeling" languages). The $IDEF1X_{97}$ identity-style language represents concepts in a more natural way by integrating data and process and by hiding implementation details that sometimes become a barrier to specifying the requirements. Hiding the implementation detail (encapsulation) simplifies the development and maintenance of databases and software.

Encapsulation is enabled by the concept that a class instance has responsibilities (see Clause 6) specified in two parts: interface and realization. By revealing only the interface specification (names, meanings, and signatures of responsibilities) to a client, $IDEF1X_{97}$ hides the complexity of realizations (the implementation detail) and their specified methods and representation properties. The realization is specified separately with the RCL so that database and software projects developed using $IDEF1X_{97}$ can focus on specifying the desired behavior and optimizing the messages requesting the services.

Managers should find the concept of modeling levels in Clause 8 of particular interest. Three technology-independent levels (survey, integration, and fully specified) and one technology-dependent level (implementation) are presented to help provide clear work product definition for management.

Clause 9 and Annex B will show management how older style information modeling ($IDEF1X_{93}$) can be supported and extended with features of the new object language.

One of the most powerful aspects of $IDEF1X_{97}$ models is that they are, with suitable automation support, directly executable to prove their correctness. Managers generally will not need to study the details, but should be aware of where to find them. The executable nature is enabled by the concepts discussed in detail in Clause 10, with a supporting RCL explained in detail in Clause 7.

Technical readers

Several groups will have primarily technical interests in the $IDEF1X_{97}$ standard.

- **Architects and Methodologists:** Readers in this group are often responsible for developing
  - The structure given to database and software components, their interrelationships, and the principles and guidelines governing their design and evolution over time (architecture) and
  - The routine procedures and practices used to produce precise, consistent, and repeatable deliverables at the end of each stage of the development process (methodology). Generally, this technical group uses modeling languages like $IDEF1X_{97}$ to develop architectures and methodologies to guide others in building consistently high quality products.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

— **Data Modelers, Object Modelers, Database Designers, Software Engineers, and Other Practitioners:** Readers in this group are often responsible for defining and specifying requirements, designing and developing databases and software system solutions, and then testing and implementing those solutions as quickly and efficiently as possible. Generally, this technical group uses modeling languages like $IDEF1X_{97}$ to develop models, designs, and products to define and satisfy operational requirements with the highest quality databases and software at the lowest risk and cost of maintenance.
— **Commercial Software Vendors:** This group includes companies that build software products and tools to support the other technical groups. These software products and tools include
  1) Database management systems, including relational, object-relational, and object-oriented,
  2) Languages (procedural and object oriented),
  3) Computer aided software engineering (CASE) tools, and
  4) Data dictionary/repository systems.

Each of these technical groups will be more naturally satisfied by different reading patterns. Although Clauses 1 and 4 provide an overview, readers in these groups will be most interested in the detailed technical topical discussions in the major clauses (Clauses 5 through 10) and the annexes.

Data modelers and database designers, for example, may want to know how the new language differs from the earlier versions of IDEF1X (Clause 9 and Annex B) and, perhaps, how to begin the transition to object modeling and design. Object modelers will need to understand all features and capabilities of $IDEF1X_{97}$ identity-style modeling.

Clause 5 delineates the two types of $IDEF1X_{97}$ classes (state and value) and describes the use of generalization and relationship concepts. For the data modelers, understanding how it is possible to use value classes in place of domains will be of interest. Object technicians in all technical groups will be interested in the value class approach and in the generalization and relationship concepts: variants of these concepts exist in many currently available object modeling and design tools and in commercial software.

If a data modeler does not intend to develop object models, Clause 6 will not be of any significant interest. However, all other technical readers should carefully read Clause 6 to gain a core understanding of the object constructs and the extent of their usage by $IDEF1X_{97}$. Clause 6 introduces the concepts of responsibility, interface, realizations, requests, properties, attributes, participant properties, operations, constraints, and notes. All technical readers should study the concepts of view, view level, environment, glossary, and model presented in Clause 8.

Clause 9 is intended for data modelers who want to or must continue the practice of key-style modeling. Other technical readers will have little interest in this clause unless they support the older style practices or are planning transitions from that style of practice to object-oriented technology. In these cases, a thorough reading of Clause 9 could help with planning for changes to architectures, methodologies, and commercial software products and tools.

For all technical readers, Clause 10 and its companion Clause 7 will present the precise definition of the language. These clauses will be a key area of study for tool builders.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

Contents

1. Overview ... 1

1.1 Scope ... 1
1.2 Purpose ... 2
1.3 Evolution of IDEF1X ... 3
1.4 Conformance ... 9

2. References ... 10

3. Definitions, acronyms, and abbreviations ... 10

3.1 Definitions ... 10
3.2 Abbreviations and acronyms ... 24

4. IDEF1X language overview ... 24

4.1 IDEF1X language constructs ... 25
4.2 IDEF1X notation ... 26

5. Class ... 31

5.1 Introduction ... 31
5.2 State class ... 35
5.3 Value class ... 41
5.4 Generalization ... 48
5.5 Relationship ... 57

6. Responsibility ... 65

6.1 Introduction ... 65
6.2 Request ... 70
6.3 Property ... 74
6.4 Attribute ... 87
6.5 Participant property ... 97
6.6 Operation ... 105
6.7 Constraint ... 113
6.8 Note ... 121

7. Rule and constraint language ... 123

7.1 Introduction ... 123
7.2 Realization ... 125
7.3 Message ... 127
7.4 Typing ... 130
7.5 Dynamic binding ... 133
7.6 Assignment ... 136
7.7 Propositions ... 137
7.8 Sentences ... 138
7.9 Type checking ... 140
7.10 Constraint checking ... 140
7.11 Query ... 141

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

7.12 Total ordering ... 141
7.13 Implementation-dependent ... 142
7.14 Lexical characteristics ... 142
7.15 RCL syntax ... 143

8. Model infrastructure constructs ... 146

8.1 View ... 147
8.2 Identity-style view level ... 153
8.3 Environment ... 155
8.4 Glossary ... 158
8.5 Model ... 160

9. Key-style modeling ... 162

9.1 Entity ... 163
9.2 Domain/value class ... 165
9.3 Key-style view ... 165
9.4 Attribute ... 167
9.5 Relationship ... 170
9.6 Entity generalization ... 178
9.7 Primary and alternate key ... 182
9.8 Foreign key ... 184
9.9 Common ancestor constraint ... 188
9.10 Key-style view level ... 191
9.11 Key-style glossary ... 195
9.12 Key-style notes ... 196
9.13 Key-style lexical rules ... 196

10. Formalization ... 197

10.1 Introduction ... 197
10.2 IDEFobject metamodel ... 200
10.3 Definition clausal form ... 207
10.4 Vocabulary ... 212
10.5 Axioms of base theories ... 217
10.6 Rewriting an IDEFobject view to definition clausal form ... 219
10.7 Formalization of the modeling constructs ... 232
10.8 Summary of the formal meaning of a view ... 264

Annex A (informative) Bibliography ... 265

Annex B (informative) Comparison of IDEF1X93 and IDEF1X97 constructs ... 267

Annex C (informative) Examples ... 270

Annex D (informative) Built-in classes ... 289

Annex E (informative) IEEE list of participants ... 304

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEF1X$_{97}$ *(IDEF$_{object}$)*

1. Overview

This standard describes the semantics and syntax of IDEF1X, a language used to represent a conceptual schema. Two styles of IDEF1X model are described. The *key style* is used to produce information models that represent the structure and semantics of data within an enterprise and is backward-compatible with the US government's Federal Information Processing Standard (FIPS) PUB 184*, Integration Definition for Information Modeling (IDEF1X)* [B13].[1] The *identity style* is used to produce object models that represent the knowledge, behavior, and rules of the concepts within an enterprise. It can be used as a growth path for key-style models. The identity style can, with suitable automation support, be used to develop a model that is an executable prototype of the target object-oriented system.

1.1 Scope

This standard defines the semantics and syntax of IDEF1X. It does so by defining the valid constructs of the language and specifying how they can be combined to form a valid model.

IDEFIX takes the approach that an enterprise manages what it knows about (its knowledge). Such knowledge consists of awareness about enterprise-pertinent actions, facts, and the relationships among them. In order to maximize the utility of this knowledge, it must be codified in a manner that makes its interpretation consistent. Without this guidance, the knowledge is either not understood at all or, worse, misused to draw unsupported or inappropriate conclusions. The guide to the interpretation and use of the enterprise knowledge has three components:

a) A grammar that dictates the kinds of actions, facts, and relationships that the enterprise is interested in recording,
b) Operations that can be performed on/with this knowledge to produce usable information, and
c) Rules about recorded knowledge that help the enterprise weed out conflicting statements and rules that govern the state changes that recorded knowledge can undergo.

[1] The numbers in brackets correspond to those of the bibliography items listed in Annex A.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

For example, the sentence "The chair sings the tree" is grammatically sound in English; there is a subject, a verb, and an object in the sentence. However, the sentence itself is not useful because it states something that is nonsensical. In a natural language, rules must be established that, for instance, indicate that the subject of the sentence must be capable of taking action, if the verb is an action, and of taking the particular action specified by the verb.

Such a guide to the interpretation and use of the enterprise knowledge is, itself, captured as a set of facts. This body of facts about facts, or metaknowledge, in turn needs a guide to *its* understanding and use. This goal, in a nutshell, is the scope of IDEF1X. As part of its semantics and syntax, IDEF1X establishes just what can be said about the enterprise knowledge and what sorts of conclusions can be drawn from that meta-knowledge.

This standard does not treat methodology. A methodology is an ordered process used to produce a repeatable result. An IDEF1X methodology deals with the process of creating a model using the IDEF1X language. While critical to the practitioner, such considerations are beyond the scope of this standard. Rather, the IDEF1X constructs will be presented individually, without regard for their logical sequence of use.

1.2 Purpose

This purpose of this standard is to describe the IDEF1X language in an unambiguous manner and thereby meet two important needs. First, those who develop and use IDEF1X models need a common understanding of the modeling constructs and rules. A precise definition of the meaning of the language components allows a model developed by one individual or group to be understood by another. Second, IDEF1X users must be supported in practice by automated tools that record and validate the models. Tool developers need a precise definition of the language so that their products assist users in applying the language correctly and allow exchange of models, at the semantic level, with other tools.

The purpose of IDEF1X as a modeling technique is the same as that of all modeling techniques employed in system analysis and development efforts, that is, to plan, build, or use systems and information systems in particular, it helps to understand the meaning of the concepts involved. Modeling provides a "language" for meanings and is sometimes referred to as closing the semantic gap between the concepts of the enterprise and the capabilities of the computer systems. Figure 1 summarizes the fundamental purpose of a model: to enable accurate and useful communication among users, analysts, and developers as they all reason about the same thing.

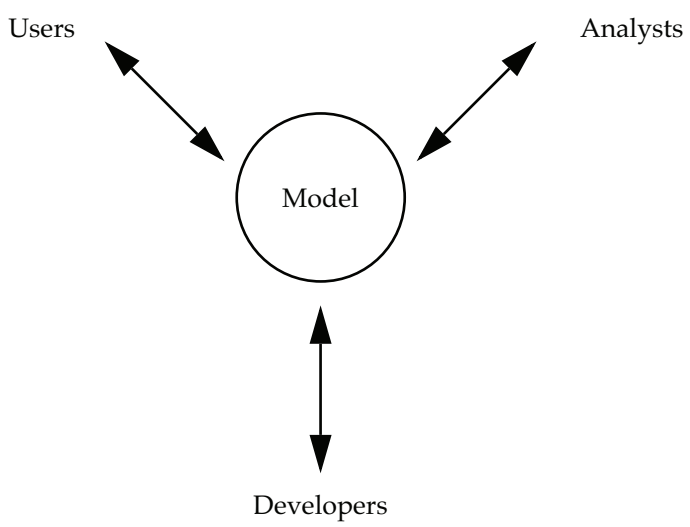

Figure 1—Communication of meanings

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

There are many uses for models, including process re-engineering, enterprise integration, detailed specification, implementation, and reverse engineering. Each is important.

1.3 Evolution of IDEF1X

The fundamental point of view originally adopted by IDEF1X was that the world was made up of interrelated things and that the meaning of data devolved from an understanding of these things and their relationships. This *key style* of IDEF1X modeling has been used over the past two decades to produce information models that represent the structure and semantics of data within an enterprise. The object model expands that point of view to include behavior. The evolution of IDEF1X has incorporated this goal of a broader understanding in the *identity-style* language introduced in this standard.

The transition from key-style to identity-style models involves bringing forward many earlier IDEF1X concepts, relaxing some of the restrictions, exploiting the fundamental concepts more fully, and adding important new concepts (see Figure 2). Each of the concepts used to produce an identity-style model is discussed fully in Clauses 4 through 8 of this standard. Clause 9 describes how to apply these concepts to produce a key-style model. Note that the concepts marked "unnecessary" in Figure 2 have been retained in the key-style language for backward-compatibility with existing models and for those who wish to continue producing key-style IDEF1X models.

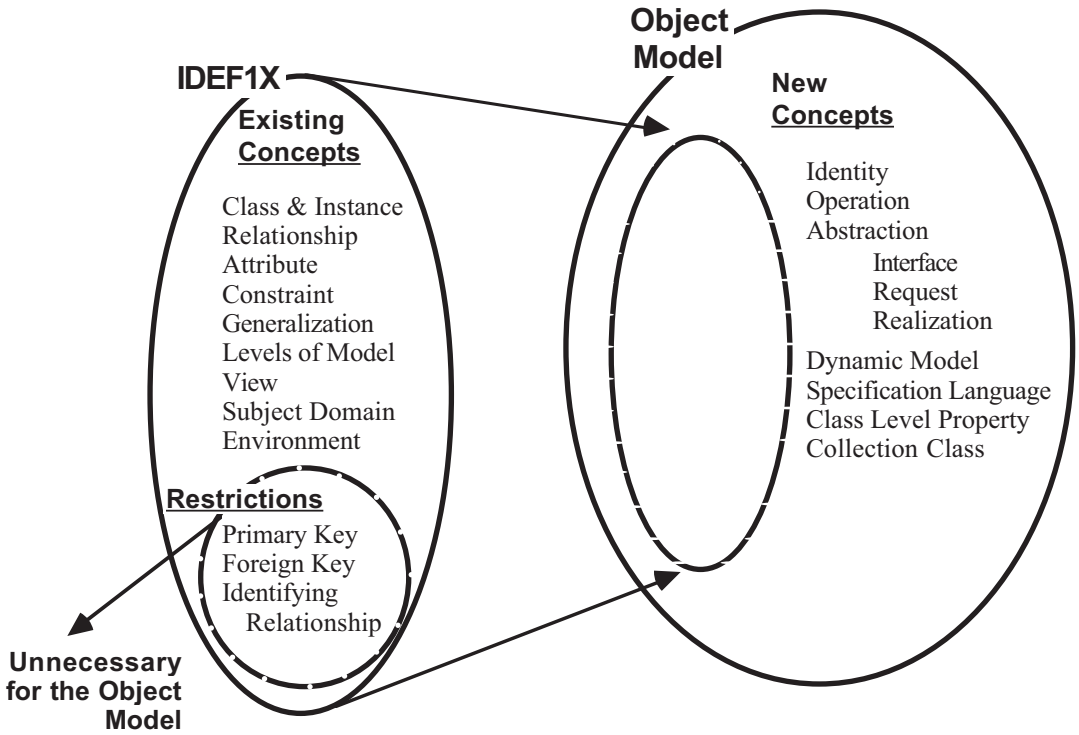

Figure 2—Correspondence of concepts

1.3.1 Understanding the data/process paradigm

The requirements for a modeling language are set largely by the way in which modelers choose to view the world. When IDEF1X was first developed in the early 1980s, the predominant system development view of

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

the world was framed in terms of data and processes. The modeling approach of this data/process (D/P) paradigm can be summarized as follows:

a) The world is made up of activities and things.
b) Things are integrated. Activities are free-standing.
c) Activities operate on things.

Within this approach, the primary objectives of an information modeling technique are

— To provide a means for understanding and analyzing an organization's data resources,
— To provide a common means of representing and reasoning about the data,
— To provide a method for presenting an overall view of the data required to run an enterprise,
— To provide a means for defining an application-independent view of data that can be validated by users and transformed into a physical database design,
— To provide a method for deriving an integrated data definition from existing data resources.

The D/P paradigm exerted a powerful and pervasive influence over all aspects of information technology. For IDEF the result was two distinct techniques: IDEF0 for process and IDEF1X for data. Thousands of successful systems have been developed using the D/P view of the world, and many developers continue to successfully employ the techniques.

1.3.2 Understanding the emerging object-oriented paradigm

The emergence of an object-oriented (OO) view of the world has strongly influenced the evolution of IDEF1X as described in this standard. The object paradigm takes a fundamentally different view of the world. In this paradigm, the modeling approach can be summarized as

a) The world is made up of objects.
b) Objects have knowledge and behavior.
c) There are no free-standing activities. Activity is accomplished by a collaboration of objects.
d) Knowledge and behavior are different aspects of the same object, considered *together,* behind an abstraction of responsibility.

Within this approach, the primary objectives of a modeling technique are

— To provide a means for understanding and analyzing the objects that are of interest to the organization,
— To provide a common means of representing and reasoning about these objects,
— To provide a method for presenting an overall view of the objects required to run an enterprise,
— To provide a means for defining an application-independent view of objects that can be validated by users and transformed into a physical design.

1.3.3 Contrasting the paradigms

The approaches of the D/P and OO paradigms are different. Major differences are summarized below in Table 1. For IDEF1X, the concepts that emerge from the D/P and OO approaches are not entirely incompatible. Indeed, there is a high degree of correspondence in the concepts.

While an IDEF1X model has typically been called a "data model," the term has always been something of a misnomer; an IDEF1X model was never a model of "data" *per se*. The entities in an IDEF1X data model are not "data" entities. An IDEF1X entity represented a concept, or meaning, in the minds of the people of the enterprise. To emphasize their concern with meaning as opposed to representational issues, "data models" like IDEF1X models are often called "semantic data models" or "conceptual models."

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

Table 1—D/P and OO approaches

D/P paradigm assumptions	OO paradigm assumptions	Contrast
An entity instance is a person, place, or thing (etc.) about which the enterprise needs to keep data.	An object is a distinct thing whose knowledge (data) or behaviors (processes) are relevant.	An object combines data and process (knowledge and behavior) and hides them behind an abstraction of responsibility.
There is no free-standing data. All data is organized around the shared real-world entities of the enterprise. The data is accessed by processes and shared across applications.	There is no free-standing knowledge (data). All knowledge is organized around the shared real-world objects of the enterprise. Each object maintains its own knowledge. The knowledge is available to (modifiable by) other objects upon request, across applications.	In D/P, processes directly access and change the data of an entity. In OO, an object must be asked for its knowledge; that knowledge is not directly accessible. Only the object itself can change its knowledge. Whether the object's knowledge is by memory or derivation is known only to the object.
Processes are free-standing. Process is organized around function, accesses entities, and is unique to an application.	There are no free-standing behaviors (processes). All behavior is organized around the shared real-world objects of the enterprise. Behavior is the responsibility of the object and available to other objects upon request, across applications.	In OO, all processing is accomplished by the actions of objects. An object acts by exploiting the knowledge and behavior of itself and collaborating objects via requests. Exactly what requests are made is known only to the object.
Similar entity instances are classified into classes, and classes are related by aggregation and generalization.	Similar objects (instances) are classified into classes, and classes are related by aggregation and generalization.	Essentially the same idea, except that the object class includes behaviors.
Each entity instance in a class is distinguishable from all others by its data values.	Each object is distinct from all other objects—it has an intrinsic, immutable identity, independent of its knowledge, behaviors, or class.	The OO model can recognize as distinct what the D/P paradigm treats as indistinguishable.
There are constraints on data.	There are constraints on both knowledge and behavior.	More general kinds of constraints are needed by the object model.
Rules are incorporated by defining processes that support them.	Rules are incorporated by defining behaviors that support them.	The D/P and OO paradigms both could be improved here. It would be better if rules could be disentangled from behaviors.

An *object model* is similarly a model of meaning, but it is a richer model that is closer to the ideal of a conceptual model. An object model attempts to capture the meanings of the knowledge and behaviors of objects. Yet, even state-of-practice object models still fall short of the ideal. The objects modeled are more like clerks than executives—they do what they are told to do but are short on vision and initiative. Objects await instruction ("Chris, put the pencil down.") rather than possessing the ability to utilize their knowledge to exhibit unrequested behavior. Nevertheless, object models are, in many environments, proving to be a major advance over the combination of separate process models and data models.

1.3.4 Expanded understanding of requirements

IDEF1X continues to meet the same requirements today that it was originally chartered to meet. However, leveraging on the capabilities that the OO approach offers, the understanding of those requirements has expanded. The expanded requirements can be summarized in terms of the five points of the "IDEF1X approach":

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

a) *Support the development of conceptual schemas*.

The conceptual schema has been characterized as those aspects of the enterprise that are invariant across the information products of the business and implementations of the enterprise business rules (application systems and databases). Previously, this scope had been understood to include only the grammar of the data. Now the understanding of the scope of the conceptual schema can be seen to include *operations* as well as *rules*.

In addition, the scope of platforms supporting applications designed using IDEF1X has broadened. In many areas, relational database management system-based applications are slowly giving way to ones built in some form of OO environment. For IDEF1X to remain transformable into functioning systems, the OO concepts must be incorporated so that the IDEF1X language is semantically broad enough to meet the needs of its users. IDEF1X needs to provide object modeling constructs appropriate for enterprise integration—from initial survey through implementation.

b) *Be a coherent language*.

IDEF1X has a clean, coherent structure with distinct and consistent semantic concepts. Many of the IDEF1X constructs have a graphical manifestation because their semantics can be easily captured that way and the resulting diagram easily read. However, as the language has evolved, not all concepts have been forced into an iconic representation; some concepts simply cannot be easily expressed graphically in a model that remains comprehensible.

In IDEF1X, as in any language, it is important that those things that are said most often are said easily, while allowing capture of those statements that are difficult to express graphically. Some constructs are best captured in text because the semantics being represented are inherently complex. Regardless of manifestation, graphical or textual, the language as a whole remains coherent and consistent.

c) *Be teachable*.

IDEF1X data modeling has been taught and practiced for nearly two decades. The teachability of the language has always been an important consideration. IDEF1X has served well as an effective communication tool across interdisciplinary teams. This rich body of experience and familiarity will not be lost. Data models created using previous versions of IDEF1X standards will continue to be conformant under this new version in the key-style language. An upward migration path for existing IDEF1X models and skill sets is provided, and training on the newer identity-style language is expected to emerge from the marketplace.

d) *Be well-tested and proven*.

The original elements of IDEF1X were based on years of experience with predecessor techniques and have been thoroughly tested both in US government development projects and in private industry. The identity style of IDEF1X introduced in this standard has been used in a variety of industry projects. Many of the features included in this version reflect requests and suggestions from IDEF1X practitioners, while others reflect the best features of the emerging object modeling techniques.

e) *Be automatable*.

IDEF1X consists of modeling constructs that can be precisely defined. The constructs of the identity-style model provide the basis for tool support for representation and reasoning about OO conceptual models, including direct execution of the models. With the formalization of the IDEF1X language, automated reasoning about the knowledge and behavior modeled is a realistic expectation.

IEEE Std 1320.2-1998 addresses the evolutionary needs of users of earlier versions of the language. Evolution is a process of change. A new version of a “creation” emerges and becomes dominant or dies out based on its suitability to the surrounding environment. During the transition, both versions of the creation will coexist.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

So, too, both versions of IDEF1X (D/P and OO) are supported by this standard. The key style of IDEF1X is fully backward-compatible with FIPS PUB 184 [B13]; the use of the identity-style features is optional. Users can migrate as needed to the expanded semantic scope characteristic of the identity-style language.

1.3.5 IDEF1X in transition

The version of IDEF1X presented in this standard is based on the object model, which is the result of the confluence of three major branches of computer science: programming, database, and artificial intelligence. As of the mid-1990s, there was no single, authoritative source for what constitutes the object model, but there was a broad consensus on the core semantic concepts. Additional semantic concepts remain in flux, and there is little consensus on syntax and methodology.

The version of IDEF1X described in FIPS PUB 184 [B13] continues to be supported by this standard and is referred to here as $IDEF1X_{93}$. Where necessary to distinguish it from this earlier version, the extended IDEF1X defined in this standard (including both identity style and key style) is referred to as $IDEF1X_{97}$.

The constructs of $IDEF1X_{97}$ were developed by

a) Framing them in terms of organizing concepts congruent with the way people think,
b) Formalizing those concepts by assigning to each a mathematical construct such that formal operations on the constructs parallel correct reasoning about the concepts,
c) Specifying a notation (diagrams or language) that actively supports representation, communication, and reasoning in terms of the concepts.

The similarities between $IDEF1X_{93}$ and $IDEF1X_{97}$ are fundamental. For both, the world consists of distinct, individual things that exist in classes[2] and are related to one another.

$IDEF1X_{97}$ was developed by relaxing some of the restrictions in $IDEF1X_{93}$, exploiting the fundamental concepts more fully, and adding some important new concepts. Each of the semantic concepts of $IDEF1X_{93}$ has a corresponding identity-style $IDEF1X_{97}$ concept, but some of the $IDEF1X_{93}$ restrictions are not needed in identity-style $IDEF1X_{97}$. These restrictions are not basically in conflict with identity-style $IDEF1X_{97}$—they could be stated if there were any reason to do so. The goals and concepts of $IDEF1X_{93}$ are subsumed by $IDEF1X_{97}$; the essential semantic constructs of $IDEF1X_{93}$ are part of $IDEF1X_{97}$. Identity-style $IDEF1X_{97}$ includes concepts that are not present in $IDEF1X_{93}$.

The identity-style $IDEF1X_{97}$ concepts are object model concepts. $IDEF1X_{97}$ includes constructs for the distinct but related components of object abstraction—interface, request, and realization. Some of the specific concepts of $IDEF1X_{97}$ that support abstraction are the principle of substitutability, declarative constraints, and declarative specifications of properties.

The identity-style $IDEF1X_{97}$ constructs model objects over varying scopes and levels of refinement. $IDEF1X_{97}$ uses both graphics and a textual specification language. Its constructs are integrated with one another by a consistent, declarative approach to object semantics.

1.3.6 Future direction

The scope of this version of the IDEF1X language covers semantic data and object modeling. Use of this standard permits the construction of data and object models that may serve to support the management of concepts as a resource, the integration of information systems, and the building of computer databases and systems.

[2]Where some say “class” and “class instance” (or, “object”), this standard adopts the terminology “class” and “instance.”

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

1.3.6.1 Topics for future extensions

Aspects of the object model that are topics for future extensions of IDEF1X include the following:

a) *Dynamic models.* This version of IDEF1X covers the specification of both the interface and realization of active properties (operations) of a class. However, this version of this standard does not provide a set of graphics describing individual requests or patterns of requests.
b) *Transaction models.* There are many transaction models, and this version of IDEF1X has chosen not to select one but rather provide only the most basic notions of stating a constraint and providing a way to check it. Future versions of this standard may expand on the treatment of "transaction."
c) *Exception handling.* The specification of exception handling is an important aspect of many object languages. Future versions of this standard may incorporate exception handling into the language.

1.3.6.2 Features for expanded scope

In addition, the scope of the language may be expanded to include coverage for features frequently requested by IDEF1X users. Typical examples include

a) *Rules beyond constraints.* "Rule" is a more general, and more powerful, idea than constraint. This version of the standard deals only with constraints. A future version could incorporate a fuller treatment of "rules."
b) *Technology-dependent levels/default transformations.* An important characteristic of the original IDEF1X was the existence of a default transformation from a fully attributed model to an implementation in a database system such as IMS™, IDMS™, xBase, or relational.[3] In addition to database and object database transforms, the expanded coverage of IDEF1X suggests transforms into popular object languages such as Smalltalk™, C++, and Java™. From the overall management and development point of view, providing transforms into technology-specific models encouraged building models that are actually used. It created a very useful "practical" counterbalance to "wishful modeling." Enterprise integration does not come about because of modeling *per se*—the models have to be used. The existence of default transformations encourages use.

1.3.6.3 Constructs for future versions

Specific constructs to be incorporated into future versions include the following:

a) *Importing concepts.* Allow importing a concept defined in one environment into another environment.
b) *Importing types.* Allow importing a type defined in one view into another view.
c) *Initial values.* Allow the specification of initial values for instance-level and class-level attributes.
d) *Interfaces.* A *class* consists of an interface, which is a set of responsibilities, and a realization for each of those responsibilities. An *interface* consists of just a set of public responsibilities and, if specified independently, can be realized by many classes. A *type* is either an interface or a class. Add support for interfaces and types as distinct from classes.
e) *Ordered relationships.* Support the specification of the ordering of instances participating in relationships.
f) *References.* Support one-way mappings to state classes in a way that is symmetric with attribute and relationship mappings.
g) *Visibility.* Support the specification of the visibility of types and their responsibilities outside their defining view.

[3]All trade or product names are either trademarks or registered trademarks of their respective companies and are the property of their respective holders. The mention of a product in this document is for the convenience of users of this standard and does not constitute an endorsement by the IEEE of these products.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

1.4 Conformance

This document is structured to permit its use in checking a model or modeling tool for conformance to this standard.

1.4.1 Identity-style model conformance

An identity-style model is conforming when

a) The lexical rules conform to Clause 4,
b) The class (state and value), generalization, and relationship semantics, syntax and rules conform to Clause 5,
c) The class (state and value), generalization, and relationship semantics conform to the semantics defined in Clause 10,
d) The responsibility semantics, syntax, rules, requests, and realizations conform to Clause 6,
e) The responsibility and realization semantics conform to the semantics defined in Clause 10,
f) The RCL conforms to the language syntax in Clause 7,
g) The RCL semantics conform to the semantics defined in Clause 10,
h) The model infrastructure constructs conform to Clause 8, and
i) The model instantiates the language metamodel in Clause 10.

1.4.2 Identity-style modeling tool conformance

An identity-style modeling tool is conforming when

a) The lexical rules conform to Clause 4,
b) The class (state and value), generalization, and relationship semantics, syntax and rules conform to Clause 5,
c) The class (state and value), generalization, and relationship semantics conform to the semantics defined in Clause 10,
d) The responsibility semantics, syntax, rules, requests, and realizations conform to Clause 6,
e) The responsibility and realization semantics conform to the semantics defined in Clause 10,
f) The RCL semantics conform to the semantics defined in Clause 10,
g) The model infrastructure constructs conform to Clause 8,
h) It can be demonstrated that the tool's metamodel maps to the language metamodel in Clause 10, that is,
    1) There is an *onto* mapping `lang` from the set of valid populations of the tool's metamodel to the set of valid populations of the language metamodel in Clause 10,
    2) There is a total mapping `tool` from the set of valid populations of the language metamodel in Clause 10 to the set of valid populations of the tool's metamodel, and
    3) For every valid population `L` of the language metamodel in Clause 10, `L = lang(tool(L))`,
i) It can be demonstrated that the tool correctly interprets RCL as specified in Clauses 7 and 10, and
j) Any tool extensions to the graphics or RCL can be demonstrated to be reducible to the graphics or RCL specified in this standard.

1.4.3 Key-style model conformance

A key-style model is conforming when

a) The lexical rules conform to Clause 4, and
b) The model components, semantics, syntax, and rules conform to Clause 8.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Full document details: ISO/IEC/IEEE 31320-2-2012

1.4.4 Key-style modeling tool conformance

A key-style modeling tool is conforming when

a) The lexical rules conform to Clause 4, and
b) The model components, semantics, syntax, and rules conform to Clause 8.

2. References

This standard should be used in conjunction with the following publication. When the cited standard is superseded by an approved revision, the revision shall apply.

IEEE Std 100-1996, IEEE Standard Dictionary of Electrical and Electronics Terms.[4]

3. Definitions, acronyms, and abbreviations

3.1 Definitions

Throughout this standard, English words are used in accordance with their definitions in the latest edition of *Webster's New Collegiate Dictionary* [B26]. Technical terms not defined in *Webster's New Collegiate Dictionary* are used in accordance with their definitions in IEEE Std 100-1996. Where a definition in IEEE Std 100-1996 does not reflect usage specific to this document, or if a term used is not defined in IEEE Std 100-1996, then an appropriate definition is provided in this clause. In some cases, a term defined in IEEE Std 100-1996 is restated in this clause where it is felt that doing so enhances the usefulness of this document. Where a term applies only to the key style of modeling, it has been annotated as such.

3.1.1 abstract class: A class that cannot be instantiated independently, i.e., instantiation must be accomplished via a subclass. A class for which every instance must also be an instance of a subclass in the cluster (i.e., a total cluster) is called an abstract class with respect to that cluster.

3.1.2 abstract data type: A data type for which the user can create instances and operate on those instances, but the range of valid operations available to the user does not depend in any way on the internal representation of the instances or the way in which the operations are realized. The data is "abstract" in the sense that values in the extent, i.e., the concrete values that represent the instances, are any set of values that support the operations and are irrelevant to the user. An abstract data type defines the operations on the data as part of the definition of the data and separates what can be done (interface) from how it is done (realization).

3.1.3 aggregate responsibility: A broadly stated responsibility that is eventually refined as specific properties and constraints.

3.1.4 alias: An alternate name for an IDEF1X model construct (class, responsibility, entity, or domain).

3.1.5 alternate key: Any candidate key of an entity other than the primary key. [key style]

3.1.6 ancestor (of a class): A generic ancestor of the class or a parent of the class or an ancestor of a parent of the class. *Contrast:* **generic ancestor; reflexive ancestor**.

3.1.7 associative class: A class introduced to resolve a many-to-many relationship.

[4] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331 USA (http://standards.ieee.org/).

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.8 associative literal: A literal that denotes an instance in terms of its value. The form of expression used to state an associative literal is ``className with propertyName: propertyValue``.

3.1.9 attribute: (A) A kind of property associated with a set of real or abstract things (people, objects, places, events, ideas, combinations of things, etc.) that is some characteristic of interest. An attribute expresses some characteristic that is generally common to the instances of a class. **(B)** An attribute is a function from the instances of a class to the instances of the value class of the attribute. **(C)** The name of the attribute is the name of the role that the value class plays in describing the class, which may simply be the name of the value class (as long as using the value class name does not cause ambiguity).

3.1.10 attribute name: A role name for the value class of the attribute.

3.1.11 bag: A kind of collection class whose members are unordered but in which duplicates are meaningful. *Contrast:* **list**; **set**.

3.1.12 behavior: The aspect of an instance's specification that is determined by the state-changing operations it can perform.

3.1.13 built-in class: A class that is a primitive in the IDEF1X metamodel.

3.1.14 candidate key: An attribute, or combination of attributes, of an entity for which no two instances agree on the values. [key style]

3.1.15 cardinality: A specification of how many instances of a first class may or must exist for each instance of a second (not necessarily distinct) class, and how many instances of a second class may or must exist for each instance of a first class. For each direction of a relationship, the cardinality can be constrained. *See also:* **cardinality constraint**.

3.1.16 cardinality constraint: (A) A kind of constraint that limits the number of instances that can be associated with each other in a relationship. *See also:* **cardinality**. **(B)** A kind of constraint that limits the number of members in a collection. *See also:* **collection cardinality**.

3.1.17 cast: To treat an object of one type as an object of another type. *Contrast:* **coerce**.

3.1.18 categorization: *See:* **generalization**. [key style]

3.1.19 category cluster: *See:* **subclass cluster**. [key style]

3.1.20 category discriminator: *See:* **discriminator**. [key style]

3.1.21 category entity: An entity whose instances represent a subtype or subclassification of another entity (generic entity). *Syn:* subclass; subtype. [key style]

3.1.22 child entity: The entity in a specific relationship whose instances can be related to zero or one instance of the other entity (parent entity). [key style]

3.1.23 class: An abstraction of the knowledge and behavior of a set of similar things. Classes are used to represent the notion of "things whose knowledge or actions are relevant."

3.1.24 class-level attribute: A mapping from the class itself to the instances of a value class.

3.1.25 class-level operation: A mapping from the (cross product of the) class itself and the instances of the input argument types to the (cross product of the) instances of the other (output) argument types.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.26 class-level responsibility: A kind of responsibility that represents some aspect of the knowledge, behavior, or rules of the class as a whole. For example, the total `registeredVoterCount` would be a class-level property of the class `registeredVoter`; there would be only one value of `registeredVoterCount` for the class as a whole. *Contrast:* **instance-level responsibility**.

3.1.27 cluster: *See:* **subclass cluster**.

3.1.28 coerce: To treat an object of one type as an object of another type by using a different object. *Contrast:* **cast**.

3.1.29 collaboration: The cooperative exchange of requests among classes and instances in order to achieve some goal.

3.1.30 collection cardinality: A specification, for a collection-valued property, of how many members the value of the property, i.e., the collection, may or must have for each instance. *See also:* **cardinality constraint**.

3.1.31 collection class: A kind of class in which each instance is a group of instances of other classes.

3.1.32 collection property: *See:* **collection-valued property**.

3.1.33 collection-valued: A value that is complex, i.e., having constituent parts. *Contrast:* **scalar**.

3.1.34 collection-valued class: A class in which each instance is a collection of values. *Contrast:* **scalar-valued class**.

3.1.35 collection-valued property: A property that maps to a collection class. *Contrast:* **scalar-valued property**.

3.1.36 common ancestor constraint: A kind of constraint that involves two or more relationship paths to the same ancestor class and states either that a descendent instance must be related to the *same* ancestor instance through each path or that it must be related to a *different* ancestor instance through each path.

3.1.37 complete cluster: *See:* **total cluster**. *Contrast:* **incomplete cluster**.

3.1.38 composite key: A key comprising of two or more attributes. [key style]

3.1.39 conceptual model: A model of the concepts relevant to some endeavor.

3.1.40 constant: (A) (As a noun) An instance whose identity is known at the time of writing. The identity of a constant state class instance is represented by #K, where K is an integer or a name. **(B)** (As an adjective) The specification that an attribute or participant property value, once assigned, may not be changed, or that an operation shall always provide the same output argument values given the same input argument values.

3.1.41 constraint: (A) A kind of **responsibility** that is a statement of facts that are required to be true in order for the constraint to be met. Classes have constraints, expressed in the form of logical sentences about property values. An instance conforms to the constraint if the logical sentence is true. Some constraints are inherent in the modeling constructs; other constraints are specific to a particular model and are stated in the specification language. **(B)** A rule that specifies a valid condition of data. [key style]

3.1.42 contravariance: A rule governing the overriding of a property and requiring that the set of values acceptable for an input argument in the overriding property shall be a superset (includes the same set) of the set of values acceptable for that input argument in the overridden property, and the set of values acceptable

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

for an output argument in the overriding property shall be a subset (includes the same set) of the set of values acceptable for that output argument in the overridden property.

3.1.43 current extent: *See:* **extensional set**.

3.1.44 data model: A graphical and textual representation of analysis that identifies the data needed by an organization to achieve its mission, functions, goals, objectives, and strategies and to manage and rate the organization. A data model identifies the entities, domains (attributes), and relationships (associations) with other data and provides the conceptual view of the data and the relationships among data. [key style]

3.1.45 data type: (A) A categorization of an abstract set of possible values, characteristics, and set of operations for an attribute. Integers, real numbers, and character strings are examples of data types. [key style] **(B)** A set of values and operations on those values. The set of values is called the *extent* of the type. Each member of the set is called an instance of the type.

3.1.46 dependent entity: An entity for which the unique identification of an instance depends upon its relationship to another entity. Expressed in terms of the foreign key, an entity is said to be dependent if any foreign key is wholly contained in its primary key. *Syn:* identifier-dependent entity. *Contrast:* **independent entity**. [key style]

3.1.47 dependent state class: A class whose instances are, by their very nature, intrinsically related to certain other state class instance(s). It would not be appropriate to have a dependent state class instance by itself and unrelated to an instance of another class(es) and, furthermore, it makes no sense to change the instance(s) to which it relates. *Contrast:* **independent state class**.

3.1.48 derived attribute: *See:* **derived property**.

3.1.49 derived participant property: *See:* **derived property**; **participant property**.

3.1.50 derived property: The designation given to a property whose value is determined by computation. The typical case of a derived property is as a derived attribute although there is nothing to prohibit other kinds of derived property.

3.1.51 discriminator: (A) A property of a superclass, associated with a cluster of that superclass, whose value identifies to which subclass a specific instance belongs. Since the value of the discriminator (when a discriminator has been declared) is equivalent to the identity of the subclass to which the instance belongs, there is no requirement for a discriminator in identity-style modeling. **(B)** An attribute in the generic entity (or a generic ancestor entity) of a category cluster whose values indicate which category entity in the category cluster contains a specific instance of the generic entity. All instances of the generic entity with the same discriminator value are instances of the same category entity. [key style]

3.1.52 domain: *Syn:* value class.

3.1.53 dynamic model: A kind of model that describes individual requests or patterns of requests among objects. *Contrast:* **static model**.

3.1.54 encapsulation: The concept that access to the names, meanings, and values of the responsibilities of a class is entirely separated from access to their realization.

3.1.55 entity: (A) The representation of a concept, or meaning, in the minds of the people of the enterprise. **(B)** The representation of a set of real or abstract things (people, objects, places, events, ideas, combination of things, etc.) that are recognized as the same type because they share the same characteristics and can participate in the same relationships. [key style]

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.56 entity instance: One of a set of real or abstract things represented by an entity. Each instance of an entity can be specifically identified by the value of the attribute(s) participating in its primary key. [key style]

3.1.57 environment: A concept space, i.e., an area in which a concept has an agreed-to meaning and one or more agreed-to names that are used for the concept.

3.1.58 environment glossary: *See:* **glossary**.

3.1.59 existence constraint: A kind of constraint stating that an instance of one entity cannot exist unless an instance of another related entity also exists. [key style]

3.1.60 existence dependency: A kind of constraint between two related entities indicating that no instance of one can exist without being related to an instance of the other. The following association types represent existence dependencies: identifying relationships, categorization structures and mandatory nonidentifying relationships. [key style]

3.1.61 extensional set: The set containing the currently existing instances of a class. The instances in the extensional set correspond to the database and data modeling notion of *instance*. *Syn:* current extent.

3.1.62 foreign key: An attribute, or combination of attributes, of a child or category entity instance whose values match those in the primary key of a related parent or generic entity instance. A foreign key results from the migration of the parent or generic entity's primary key through a generalization structure or a relationship. [key style]

3.1.63 formalization: The precise description of the semantics of a language in terms of a formal language such as first order logic.

3.1.64 framework: A reusable design (models and/or code) that can be refined (specialized) and extended to provide some portion of the overall functionality of many applications.

3.1.65 function: A single-valued mapping. The mapping `M` from `D` to `R` is a *function* if for any `X` in `D` and `Y` in `R`, there is at most one pair `[ X, Y ]` in `M`. *Syn:* single-valued. *Contrast:* **multi-valued**.

3.1.66 generalization: (A) Saying that a subclass `S` generalizes to a superclass `C` means that every instance of class `S` is also an instance of class `C`. Generalization is fundamentally different from a *relationship*, which <u>may</u> associate distinct instances. **(B)** A taxonomy in which instances of both entities represent the same real or abstract thing. One entity (the generic entity) represents the complete set of things and the other (category entity) represents a subtype or sub-classification of those things. The category entity may have one or more attributes, or relationships with instances of another entity, not shared by all generic entity instances. Each instance of the category entity is simultaneously an instance of the generic entity. [key style]

3.1.67 generalization hierarchy: *See:* **generalization taxonomy**.

3.1.68 generalization network: *See:* **generalization taxonomy**.

3.1.69 generalization structure: A connection between a superclass and one of its more specific, immediate subclasses.

3.1.70 generalization taxonomy: A set of generalization structures with a common generic ancestor. In a generalization taxonomy every instance is fully described by one or more of the classes in the taxonomy. The structuring of classes as a generalization taxonomy determines the inheritance of responsibilities among classes.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.71 generic ancestor (of a class): A superclass that is either an immediate superclass of the class or a generic ancestor of one of the superclasses of the class. *Contrast:* **ancestor**. *See also:* **reflexive ancestor**.

3.1.72 generic entity: An entity whose instances are classified into one or more subtypes or subclassifications (category entities). *Syn:* superclass; supertype. [key style]

3.1.73 glossary: The collection of the names and narrative descriptions of all terms that may be used for defined concepts (views, classes, subject domains, relationships, responsibilities, properties, and constraints) within an environment.

3.1.74 hidden: A general term covering both private and protected. *Contrast:* **public**. *See also:* **private**; **protected**.

3.1.75 IDEF1X model: A set of one or more IDEF1X views, often represented as view diagrams that depict the underlying semantics of the views, along with definitions of the concepts used in the views.

3.1.76 identifier dependency: A kind of constraint between two related entities requiring the primary key in one (child entity) to contain the entire primary key of the other (parent entity). Identifying relationships and categorization structures represent identifier dependencies. [key style]

3.1.77 identifier-dependent entity: *Syn:* dependent entity.

3.1.78 identifier-independent entity: *Syn:* independent entity.

3.1.79 identifying relationship: A kind of specific (not many-to-many) relationship in which every attribute in the primary key of the parent entity is contained in the primary key of the child entity. *Contrast:* **nonidentifying relationship**. [key style]

3.1.80 identity: The inherent property of an instance that distinguishes it from all other instances. Identity is intrinsic to the instance and independent of the instance's property values or the classes to which the instance belongs.

3.1.81 identity-style view: A view produced using the identity-style modeling constructs.

3.1.82 immutable class: A class for which the set of instances is fixed; its instances do not come and go over time. *Contrast:* **mutable class**. *See also:* **value class**.

3.1.83 incomplete cluster: *See:* **partial cluster**. *Contrast:* **complete cluster**.

3.1.84 independent entity: An entity for which each instance can be uniquely identified without determining its relationship to another entity. *Syn:* identifier-independent entity. *Contrast:* **dependent entity**. [key style]

3.1.85 independent state class: A state class that is not a dependent state class. *Contrast:* **dependent state class**.

3.1.86 inheritance: A semantic notion by which the responsibilities (properties and constraints) of a subclass are considered to include the responsibilities of a superclass, in addition to its own, specifically declared responsibilities.

3.1.87 inherited attribute: (A) An attribute that is a characteristic of a class by virtue of being an attribute of a generic ancestor. **(B)** An attribute that is a characteristic of a category entity by virtue of being an attribute in its generic entity or a generic ancestor entity. [key style]

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.88 input argument: The designation given to an operation argument that will always have a value at the invocation of the operation. *Contrast:* **output argument**.

3.1.89 instance: A discrete, bounded thing with an intrinsic, immutable, and unique identity. Anything that is classified into a class is said to be an instance of the class. All the instances of a given class have the same responsibilities, i.e., they possess the same kinds of knowledge, exhibit the same kinds of behavior, participate in the same kinds of relationships, and obey the same rules. Unless otherwise noted, *instance* means an existing instance, that is, a member of the current extent.

3.1.90 instance-level attribute: A mapping from the instances of a class to the instances of a value class.

3.1.91 instance-level operation: A mapping from the (cross product of the) instances of the class and the instances of the input argument types to the (cross product of the) instances of the other (output) argument types.

3.1.92 instance-level responsibility: A kind of responsibility that applies to each instance of the class individually. *Contrast:* **class-level responsibility**.

3.1.93 interface: The declaration of the meaning and the signature for a property or constraint. The interface states "what" a property (responsibility) knows or does or what a constraint (responsibility) must adhere to. The interface specification consists of the meaning (semantics) and the signature (syntax) of a property or constraint.

3.1.94 intrinsic: The specification that a property is total (i.e., mandatory), single-valued, and constant.

3.1.95 intrinsic relationship: A kind of relationship that is total, single-valued, and constant from the perspective of (at least) one of the participating classes, referred to as a *dependent class*. Such a relationship is considered to be an integral part of the essence of the dependent class. For example, a transaction has an intrinsic relationship to its related account because it makes no sense for an instance of a transaction to "switch" to a different account since that would change the very nature of the transaction. *Contrast:* **nonintrinsic relationship**.

3.1.96 key migration: The modeling process of placing the primary key of a parent or generic entity in its child or category entity as a foreign key. [key style]

3.1.97 key-style view: A view that represents the structure and semantics of data within an enterprise, i.e., data (information) models. The key-style view is backward-compatible with FIPS PUB 184 [B13].

3.1.98 knowledge: The aspect of an instance's specification that is determined by the values of its attributes, participant properties, and constant, read-only operations.

3.1.99 label: A word or phrase that is attached to or part of a model graphic. A label typically consists of a model construct's name (or one of the aliases) and may contain additional textual annotations (such as a note identifier).

3.1.100 level: A designation of the coverage and detail of a view. There are multiple levels of view; each is intended to be distinct, specified in terms of the modeling constructs to be used.

3.1.101 list: A kind of collection class that contains no duplicates and whose members are ordered. *Contrast:* **bag**; **set**.

3.1.102 literal: The denotation of the specific instance of a value class.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.103 lowclass: If an instance is in a class `S` and not in any subclass of `S`, then `S` is the lowclass for the instance.

3.1.104 mandatory: A syntax keyword used to specify a total mapping. *Contrast:* **optional**. *See also:* **total**.

3.1.105 mandatory nonidentifying relationship: A kind of nonidentifying relationship in which an instance of the child entity must be related to an instance of the parent entity. *Contrast:* **optional nonidentifying relationship**. *See also:* **nonidentifying relationship**. [key style]

3.1.106 many-to-many relationship: A kind of relationship between two state classes (not necessarily distinct) in which each instance of one class may be associated with any number of instances of a second class (possibly none), and each instance of the second class may be related to any number of instances of the first class (possibly none).

3.1.107 mapping: An assigned correspondence between two things that is represented as a set of ordered pairs. Specifically, a mapping from a class to a value class is an attribute. A mapping from a state class to a state class is a participant property. A mapping from the (cross product of the) instances of the class and the instances of the input argument types to the (cross product of the) instances of the other (output) argument types is an operation.

3.1.108 mapping completeness: A designation of whether a mapping is complete (totally mapped) or incomplete (partial). *See also:* **partial**; **total**.

3.1.109 meaning: (of a responsibility) A statement of what the responsibility means. The statement of responsibility is written from the point of view of the requester, not the implementer. The statement of responsibility states what the requester needs to know to make intelligent use of the property or constraint. That statement should be complete enough to let a requester decide whether to make the request, but it should stop short of explaining how a behavior or value is accomplished or derived. Meaning is initially captured using freeform natural language text in a glossary definition. It may be more formally refined into a statement of *pre-conditions* and *post-conditions* using the *specification language*.

3.1.110 message: A communication sent from one object to another. *Message* encompasses requests to meet responsibilities as well as simple informative communications. *See also:* **request**.

3.1.111 metamodel: A metamodel `Vm` for a subset of IDEF_{object} is a view of the constructs in the subset that is expressed using those constructs such that there exists a valid instance of `Vm` that is a description of `Vm` itself.

3.1.112 method: A statement of how property values are combined to yield a result.

3.1.113 migrated attribute: A foreign key attribute of a child entity. [key style]

3.1.114 migrated key: *Syn:* foreign key. [key style]

3.1.115 model: (A) A representation of something that suppresses certain aspects of the modeled subject. This suppression is done in order to make the model easier to deal with and more economical to manipulate and to focus attention on aspects of the modeled subject that are important for the intended purpose of the model. For instance, an accurate model of the solar system could be used to predict when planetary conjunctions will take place and the phases of the moon at a particular time. Such a model would generally not attempt to represent the internal workings of the sun or the surface composition of each planet. **(B)** An interpretation of a theory for which all the axioms of the theory are true. [logic sense]

3.1.116 model glossary: The collection of the names and definitions of all defined concepts that appear within the views of a model.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.117 multi-valued: A mapping that is not a function. *Contrast:* **function**; **single-valued**.

3.1.118 multi-valued property: A property with a multi-valued mapping. *Contrast:* **single-valued property**.

3.1.119 multiple inheritance: The ability of a subclass to inherit responsibilities from more than one superclass.

3.1.120 mutable class: A class for which the set of instances is not fixed; its instances come and go over time. *Contrast:* **immutable class**. *See also:* **state class**.

3.1.121 name: A word or phrase that designates some model construct (such as a class, responsibility, subject domain, etc.).

3.1.122 named constraint: A constraint that is specific to a particular model, rather than being inherent in some modeling construct (such as a cardinality constraint.). A named constraint is explicitly named, its meaning is stated in natural language, and its realization is written in the specification language.

3.1.123 nonidentifying relationship: A kind of specific (not many-to-many) relationship in which some or all of the attributes contained in the primary key of the parent entity do not participate in the primary key of the child entity. *Contrast:* **identifying relationship**. *See also:* **mandatory nonidentifying relationship**, **optional nonidentifying relationship**. [key style]

3.1.124 nonintrinsic relationship: A kind of relationship that is partial, is multi-valued, or may change. *Contrast:* **intrinsic relationship**.

3.1.125 nonkey attribute: An attribute that is not the primary or a part of a composite primary key of an entity. [key style]

3.1.126 note: A body of free text that describes some general comment or specific constraint about a portion of a model. A note may be used in an early, high-level view prior to capturing constraints in the specification language; a note may further clarify a rule by providing explanations and examples. A note may also be used for "general interest" comments not involving rules. These notes may accompany the model graphics.

3.1.127 object: *Syn:* instance.

3.1.128 object identifier: Some concrete representation for the identity of an object (instance). The object identifier (oid) is used to show examples of instances with identity, to formalize the notion of identity, and to support the notion in programming languages or database systems.

3.1.129 object model: An integrated abstraction that treats all activities as performed by collaborating objects and encompassing both the data and the operations that can be performed against that data. An object model captures both the meanings of the knowledge and actions of objects behind the abstraction of responsibility.

3.1.130 oid: *See:* **object identifier**.

3.1.131 one-to-many relationship: A kind of relationship between two state classes in which each instance of one class, referred to as the *child* class, is specifically constrained to relate to no more than one instance of a second class, referred to as the *parent* class.

3.1.132 operation: A kind of property that is a mapping from the (cross product of the) instances of the class and the input argument types to the (cross product of the) instances of the other (output) argument types. The operations of a class specify the behavior of its instances. While an attribute or participant property is an

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

abstraction of what an instance knows, an operation is an abstraction of what an instance does. Operations can perform input and output, and can change attribute and participant property values. Every operation is associated with one class and is thought of as a responsibility of that class. No operations are the joint responsibility of multiple classes.

3.1.133 optional: A syntax keyword used to specify a partial mapping. *Contrast:* **mandatory**. *See also:* **partial**.

3.1.134 optional attribute: An attribute that may have no value for an instance.

3.1.135 optional nonidentifying relationship: A kind of nonidentifying relationship in which an instance of the child entity can exist without being related to an instance of the parent entity. *Contrast:* **mandatory nonidentifying relationship**. *See also:* **nonidentifying relationship**. [key style]

3.1.136 output argument: An argument that has not been specified as an input argument. It is possible for an output argument to have no value at the time a request is made. *Contrast:* **input argument**.

3.1.137 override: The ability of a property in a subclass to respecify the realization of an inherited property of the same name while retaining the same meaning.

3.1.138 overriding property: A property in a subclass that has the same meaning and signature as a similarly named property in one of its superclasses, but has a different realization.

3.1.139 owned attribute: An attribute of an entity that has not migrated into the entity. [key style]

3.1.140 parallel classes: A pair of classes that are distinct, are not mutually exclusive and have a common generic ancestor class and for which neither is a generic ancestor of the other.

3.1.141 parameterized collection class: A kind of collection class restricted to hold only instances of a specified type (class).

3.1.142 parent entity: An entity in a specific relationship whose instances can be related to a number of instances of another entity (child entity). [key style]

3.1.143 partial: An incomplete mapping, i.e., some instances map to no related instance. An attribute may be declared partial, meaning it may have no value. A participant property is declared optional as part of the relationship syntax. An operation is declared partial when it may have no meaning for some instances, i.e., it may not give an answer or produce a response. *Contrast:* **total**. *See also:* **mapping completeness**; **optional**.

3.1.144 partial cluster: A subclass cluster in which an instance of the superclass may exist without also being an instance of any of the subclasses. *Contrast:* **total cluster**. *See also:* **superclass**.

3.1.145 participant property: A kind of property of a state class that reflects that class' knowledge of a relationship in which instances of the class participate. When a relationship exists between two state classes, each class contains a participant property for that relationship. A participant property is a mapping from a state class to a related (not necessarily distinct) state class. The name of each participant property is the name of the role that the other class plays in the relationship, or it may simply be the name of the class at the other end of the relationship (as long as using the class name does not cause ambiguity). A value of a participant property is the identity of a related instance.

3.1.146 path assertion: *See:* **common ancestor constraint**.

3.1.147 post-condition: A condition that is guaranteed to be true after a successful property request.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.148 pre-condition: A condition that is required to be true before making a property request.

3.1.149 primary key: The candidate key selected as the unique identifier of an entity. [key style]

3.1.150 private: A responsibility that is visible only to the class or the receiving instance of the class (available only within methods of the class). *Contrast:* **protected**; **public**. *See also:* **hidden**.

3.1.151 property: A kind of responsibility that is an inherent or distinctive characteristic or trait that manifests some aspect of an object's knowledge or behavior. Three kinds of property are defined: attributes, participant properties due to relationships, and operations.

3.1.152 protected: A responsibility that is visible only to the class or the receiving instance of the class (available only within methods of the class or its subclasses). *Contrast:* **private**; **public**. *See also:* **hidden**.

3.1.153 public: A responsibility that is not hidden, i.e., visible to any requester (available to all without restriction). *Contrast:* **hidden**; **private**; **protected**.

3.1.154 RCL: *See:* **Rule and Constraint Language**.

3.1.155 read-only: A property that causes no state changes, i.e., it does no updates.

3.1.156 realization: The representation of interface responsibilities through specified algorithms and any needed representation properties. The realization states "how" a responsibility is met; it is the statement of the responsibility's method. Realization consists of any necessary representation properties together with the algorithm (if any). A realization may involve representation properties or an algorithm, or both. For example, an attribute typically has only a representation and no algorithm. An algorithm that is a "pure algorithm" (i.e., without any representation properties) uses only literals; it does not "get" any values as its inputs. Finally, a derived attribute or operation typically has both an algorithm and representation properties.

3.1.157 referential integrity: (A) A guarantee that a reference refers to an object that exists. **(B)** A guarantee that all specified conditions for a relationship hold true. For example, if a class is declared to require at least one instance of a related state class, it would be invalid to allow an instance that does not have such a relationship.

3.1.158 reflexive ancestor (of a class): The class itself or any of its generic ancestors. *See also:* **generic ancestor**. *Contrast:* **ancestor**.

3.1.159 relationship: A kind of association between two (not necessarily distinct) classes that is deemed relevant within a particular scope and purpose. The association is named for the sense in which the instances are related. A relationship can be represented as a time-varying binary relation between the instances of the current extents of two state classes.

3.1.160 relationship instance: An association of specific instances of the related classes.

3.1.161 relationship name: A verb or verb phrase that reflects the meaning of the relationship expressed between the two entities shown on the diagram on which the name appears. [key style]

3.1.162 representation: One or more properties used by an algorithm for the realization of a responsibility.

3.1.163 representation property: A property on which an algorithm operates.

3.1.164 request: A message sent from one object (the sender) to another object (the receiver), directing the receiver to fulfill one of its responsibilities. Specifically, a request may be for the value of an attribute, for the value of a participant property, for the application of an operation, or for the truth of a constraint. *Request* also

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

encompasses sentences of such requests. Logical sentences about the property values and constraints of objects are used for queries, pre-conditions, post-conditions, and responsibility realizations. *See also:* **message**.

3.1.165 respecialize: A change by an instance from being an instance of its current subclass to being an instance of one of the other subclasses in its current cluster. *Contrast:* **specialize**; **unspecialize**.

3.1.166 responsibility: A generalization of properties (attributes, participant properties, and operations) and constraints. An instance possesses knowledge, exhibits behavior, and obeys rules. These are collectively referred to as the instance's responsibilities. A class abstracts the responsibilities in common to its instances. A responsibility may apply to each instance of the class (instance-level) or to the class as a whole (class-level).

3.1.167 role name: (A) A name that more specifically names the nature of a related value class or state class. For a relationship, a role name is a name given to a class in a relationship to clarify the participation of that class in the relationship, i.e., connote the role played by a related instance. For an attribute, a role name is a name used to clarify the sense of the value class in the context of the class for which it is a property. **(B)** A name assigned to a foreign key attribute to represent the use of the foreign key in the entity. [key style]

3.1.168 Rule and Constraint Language: A declarative specification language that is used to express the realization of responsibilities and to state queries.

3.1.169 sample instance diagram: A form of presenting example instances in which instances are shown as separate graphic objects. The graphic presentation of instances can be useful when only a few instances are presented. *Contrast:* **sample instance table**.

3.1.170 sample instance table: A form of presenting example instances in which instances are shown as a tabular presentation. The tabular presentation of instances can be useful when several instances of one class are to be presented. *Contrast:* **sample instance diagram**.

3.1.171 scalar: A value that is atomic, i.e., having no parts. *Contrast:* **collection-valued**.

3.1.172 scalar property: *See:* **scalar-valued property**.

3.1.173 scalar-valued class: A class in which each instance is a single value. *Contrast:* **collection-valued class**.

3.1.174 scalar-valued property: A property that maps to a scalar-valued class. *Contrast:* **collection-valued property**.

3.1.175 semantics: The meaning of the syntactic components of a language.

3.1.176 set: A kind of collection class with no duplicate members and where order is irrelevant. *Contrast:* **bag**; **list**.

3.1.177 shadow class: A class presented in a view that is specified in some other view.

3.1.178 signature: A statement of what the interface to a responsibility "looks like." A signature consists of the responsibility name, along with a property operator and the number and type of its arguments, if any. A type (class) may be specified for each argument in order to limit the argument values to being instances of that class.

3.1.179 single-valued property: A property with a single-valued mapping. *Contrast:* **multi-valued property**.

3.1.180 single-valued: ***Syn:*** function. *Contrast:* **multi-valued**.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.181 specialize: A change by an instance from being an instance of its current class to being additionally an instance of one (or more) of the subclasses of the current subclass. A specialized instance acquires a different (lower) lowclass. *Contrast:* **respecialize**; **unspecialize**.

3.1.182 specification language: *See:* **Rule and Constraint Language**.

3.1.183 split key: A foreign key containing two or more attributes, where at least one of the attributes is a part of the entity's primary key and at least one of the attributes is not a part of the primary key. [key style]

3.1.184 state class: A kind of class that represents a set of real or abstract objects (people, places, events, ideas, things, combinations of things, etc.) that have common knowledge or behavior. A state class represents instances with changeable state. The constituent instances of a state class can come and go and can change state over time, i.e., their property values can change.

3.1.185 static model: A kind of model that describes an interrelated set of classes (and/or subject domains) along with their relationships and responsibilities. *Contrast:* **dynamic model**.

3.1.186 subclass: A specialization of one or more superclasses. Each instance of a subclass is an instance of each superclass. A subclass typically specifies additional, different responsibilities to those of its superclasses or overrides superclass responsibilities to provide a different realization.

3.1.187 subclass cluster: (A) A set of one or more generalization structures in which the subclasses share the same superclass and in which an instance of the superclass is an instance of no more than one subclass. A cluster exists when an instance of the superclass can be an instance of only one of the subclasses in the set, and each instance of a subclass is an instance of the superclass. **(B)** A set of one or more mutually exclusive specializations of the same generic entity. [key style]

3.1.188 subclass responsibility: A designation that a property of a class must be overridden in its subclasses, i.e., the designation given to a property whose implementation is not specified in this class. A property that is a subclass responsibility is a specification in the superclass of an *interface* that each of its subclasses must provide. A property that is designated as a subclass responsibility has its *realization* deferred to the subclass(es) of the class.

3.1.189 subject domain: An area of interest or expertise. The responsibilities of a subject domain are an aggregation of the responsibilities of a set of current or potential named classes. A subject domain may also contain other subject domains. A subject domain encapsulates the detail of a view.

3.1.190 subject domain responsibility: A generalized concept that the analyst discovers by asking "in general, what do instances in this subject domain need to be able to do or to know?" The classes and subject domains in a subject domain together supply the knowledge, behavior, and rules that make up the subject. These notions are collectively referred to as the subject domain's responsibilities. Subject domain responsibilities are not distinguished as sub-domains or classes during the early stages of analysis.

3.1.191 substitutability: A principle stating that, since each instance of a subclass is an instance of the superclass, an instance of the subclass should be acceptable in any context where an instance of the superclass is acceptable. Any request sent to an instance receives an acceptable response, regardless of whether the receiver is an instance of the subclass or the superclass.

3.1.192 subtype: *Syn:* subclass.

3.1.193 superclass: A class whose instances are specialized into one or more subclasses. *See also:* **partial cluster**; **total cluster**.

3.1.194 supertype: *Syn:* superclass.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.195 syntax: The structural components or features of a language and rules that define the ways in which the language constructs may be assembled together to form sentences.

3.1.196 total: A complete mapping. The mapping `M` from a set `D` to a set `R` is *total* if for every `X` in `D`, there is at least one Y in R and pair `[ X, Y ]` in `M`. A property of a class is total, meaning that it will have a value for every instance of the class, unless it is explicitly declared partial. *Contrast:* **partial**. *See also:* **mandatory**; **mapping completeness**.

3.1.197 total cluster: A subclass cluster in which each instance of a superclass must be an instance of at least one of the subclasses of the cluster. *Contrast:* **partial cluster**. *See also:* **superclass**.

3.1.198 type: *See:* **class**.

3.1.199 uniqueness constraint: A kind of constraint stating that no two distinct instances of a class may agree on the values of all the properties that are named in the uniqueness constraint.

3.1.200 unspecialize: A change by an instance from being an instance of its current subclass within a cluster to being an instance of none of the subclasses in the cluster. *Contrast:* **respecialize**; **specialize**.

3.1.201 updatable argument: The designation given to an operation argument that identifies an instance to which a request may be sent that will change the state of the instance. An argument not designated as "updatable" means that there will be no requests sent that will change the state of the instance identified by the argument.

3.1.202 value class: A kind of class that represents instances that are pure values. The constituent instances of a value class do not come and go and cannot change state.

3.1.203 value list constraint: A kind of constraint that specifies the set of all acceptable instance values for a value class.

3.1.204 value range constraint: A kind of constraint that specifies the set of all acceptable instance values for a value class where the instance values are constrained by a lower and/or upper boundary. An example of the value range constraint is `Azimuth`, which is required to be between –180° to +180°. A range constraint only makes sense if there is a linear ordering specified.

3.1.205 variable: An instance whose identity is unknown at the time of writing. A variable is represented by an identifier that begins with an upper-case letter.

3.1.206 verb phrase: (A) A part of the label of a relationship that names the relationship in a way that a sentence can be formed by combining the first class name, the verb phrase, the cardinality expression, and the second class name or role name. A verb phrase is ideally stated in active voice. For example, the statement "*each project funds one or more tasks"* could be derived from a relationship showing "`project`" as the first class, "`task`" as the second class with a "one or more" cardinality, and "funds" as the verb phrase. **(B)** A phrase used to name a relationship, which consists of a verb and words that constitute the object of the phrase. [key style]

3.1.207 view: (A) A collection of subject domains, classes, relationships, responsibilities, properties, constraints, and notes assembled or created for a certain purpose and covering a certain scope. A view may cover the entire area being modeled or only a part of that area. **(B)** A collection of entities and assigned attributes (domains) assembled for some purpose. [key style]

3.1.208 view diagram: A graphic representation of the underlying semantics of a view.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

3.1.209 visibility: The specification, for a property, of "who can see it?"—i.e., whose methods can reference the property. Visibility is either private, protected, or public.

3.1.210 whitespace: The nondisplaying formatting characters such as spaces, tabs, etc., that are embedded within a block of free text.

3.2 Abbreviations and acronyms

ADT	abstract data type
BNF	Backus-Naur form
DBDG	Database Design Group
DBMS	database management system
DDL	Data Definition Language
DISA	Defense Information Systems Agency
DMSO	Defense Modeling and Simulations Office
D/P	Data/Process
ER	entity-relationship
FA	fully attributed
FIPS	Federal Information Processing Standard
GUI	Graphical User Interface
I^2S^2	Integrated Information Support System
KB	key-based
LDDT	Logical Database Design Technique
NIST	National Institute of Standards and Technology
oid	object identifier
OO	object-oriented
RCL	Rule and Constraint Language
SQL	Structured Query Language
UOD	universe of discourse

4. IDEF1X language overview

IDEF1X is a language, and like any language it has parts of speech. For example, the classes and instances are the nouns, and the relationships are roles that instances of one class may play relative to instances of another class. The responsibilities are the knowledge that the classes and instances may possess, the behaviors that the classes and instances may exhibit, and the rules that they must obey.[5] Each of these "parts of speech" has a particular meaning and, because of that meaning, each may be combined with others only in specific ways. For example, because of what they represent, it does not make sense in IDEF1X to have relationships between relationships. This standard establishes what the valid constructs are and which possible combinations of IDEF1X modeling constructs constitute a valid model.

In Clauses 5 through 6 and Clause 8, the meaning of each basic IDEF1X construct is informally described in English, the graphic syntax for the construct (where there is one) is stated and illustrated, and any rules for using the construct are listed.[6] Clause 7 provides a full specification of the RCL syntax and semantics. Clause 10 provides a formal specification of each of the IDEF1X concepts in first-order language. Every effort has been made to insure that the English explanation of these constructs is complete and accurate. However, if there appears to be an inconsistency between the English description of a construct and the formalization or the language specification of that construct, the formalization or the language specification (in that order) is the authoritative statement.

[5]Constraint is the only form of rule discussed in this version of the standard.

[6]Rules apply to a completed model (model construct) and not one that is in development.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

4.1 IDEF1X language constructs

This clause introduces the constructs of the IDEF1X language. Only a summary is given here; details and numerous examples are provided in later clauses. The language constructs of IDEF1X include

a) Class. A *class* is an abstraction of the knowledge and behavior of a set of similar things. Anything that is classified into a class is said to be an *instance* of the class. All the instances of a given class have the same *responsibilities*, i.e., they possess the same kinds of knowledge, exhibit the same kinds of behavior, and adhere to the same rules. An instance is a discrete, bounded thing with an intrinsic, immutable, and unique *identity*.

   Each class is either a state class or a value class.

   1) State Class. A *state class* represents instances with changeable state. Its instances can come and go and can change state over time, i.e., their property values can change.
   2) Value Class. A *value class* represents instances that are pure values. Its instances do not come and go and cannot change state.

b) Generalization. Classes are used to represent the notion of "things whose knowledge or actions are relevant." Since some real world things are generalizations of other real world things, some classes must, in some sense, be generalizations of other classes. A class that specifies additional, different responsibilities to those of a more general class is known as a *subclass* of that more general class (its *superclass*). Each instance of the subclass represents the same real-world thing as its instance in the superclass. The structuring of classes as a generalization taxonomy (hierarchy or network) determines the inheritance of responsibilities among classes.

c) Relationship. A *relationship* expresses a connection between two state classes that is deemed relevant for a particular scope and purpose. It is named for the sense in which the instances are related.

d) Responsibility. An instance possesses *knowledge*, exhibits *behavior*, and obeys *rules*. These notions are collectively referred to as the instance's *responsibilities*. A class abstracts the responsibilities in common to its instances. During initial model development, a responsibility may simply be stated in general terms and not distinguished explicitly as an attribute, participant property, operation, or constraint. Also, aggregate responsibilities may be specified, rather than individual properties. Broadly stated responsibilities are eventually refined as specific properties and constraints.

   1) Property. Some responsibilities are met by knowledge and behavior which, in turn, are determined by properties. A *property* is an inherent or distinctive characteristic or trait that manifests some aspect of an object's knowledge or behavior. There are three kinds of property: *attributes*, *participant properties* due to relationships, and *operations*. Classes have properties; instances have *property values*.

      i) Attribute. An *attribute* is a mapping from a class to a value class. An attribute expresses some characteristic that is generally common to the instances of a class. The name of the attribute is the name of the role that the value class plays in describing the class, which may simply be the name of the value class (as long as using the value class name does not cause ambiguity[7]).

      ii) Participant property. A *participant property* is a mapping from a state class to a related (not necessarily distinct) state class. When a relationship exists between two state classes, each class contains a participant property for that relationship. The name of each participant property is the name of the role that the other class plays in the relationship, or it may simply be the name of the class at the other end of the relationship (as long as using the class name does not cause ambiguity[8]). The value of a participant property is the identity of a related instance. For a relationship in which there may be many related instances, there is also a participant property named as described above but suffixed with `(s)`, which is a mapping from the state class to a collection class in which the members of the collection are the related instances.

[7] Ambiguity would exist if there were multiple mappings between a class and value class and different role names were not used.

[8] Ambiguity would exist if there were more than one relationship between the same pair of classes and different role names were not used.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

iii) Operation. The operations of a class specify the behavior of its instances. An *operation* is a mapping from the (cross product of the) instances of the class and the instances of the input argument types to the (cross product of the) instances of the other (output) argument types. While an attribute or participant property is an abstraction of what an instance knows, an operation is an abstraction of what an instance does.

2) Constraint. Other responsibilities are met by adhering to constraints. A *constraint* is a statement of facts that are required to be true for a class or the instances of a class. Constraints are expressed in the form of logical sentences about property values or constraints. An instance conforms to the constraint if the logical sentence is true for that instance. Some constraints are inherent in the modeling constructs and can be readily represented using the graphics; other constraints are specific to a particular model and are stated in the specification language.

3) Note. A *note* is a body of free text that describes some general comment or specific constraint about a portion of a model. A note may be used in an early, high-level view prior to capturing constraints in the specification language; a note may further clarify a rule by providing explanations and examples. A note may also be used for "general interest" comments not involving rules. These notes may accompany the model graphics.

e) Request. A request is a message sent from one object (the sender) to another object (the receiver), directing the receiver to fulfill one of its responsibilities. Specifically, a request may be for the value of an attribute, for the value of a participant property, for the application of an operation, or for the truth of a constraint.

f) Realization. The realization of a responsibility specifies how the responsibility is met. A realization is stated as a logical sentence giving the necessary and sufficient conditions that the responsibility be met.

g) Model infrastructure constructs. Modeled constructs are presented in views and packaged as models that provide supporting elements of documentation such as textual descriptions.

1) View. A *view* is a collection of subject domains, classes, relationships, responsibilities, properties, constraints, and notes (and possibly other views) assembled or created for a certain purpose and covering a certain scope. A view may cover the entire area being modeled or only a part of that area.[9]

2) Level. A *level* is a designation of the coverage and detail of a view. There are multiple levels of view.

3) Environment. An *environment* is a concept space, i.e., an area in which a concept has an agreed-to meaning and one or more agreed-to names that are used for the concept. Every view is developed for a specific environment.

4) Glossary. A *glossary* is the collection of the names and descriptions of all terms that may be used for defined concepts (views, subject domains, classes, relationships, responsibilities, properties, and constraints) within an environment. A *model glossary* is the collection of the names and descriptions of all defined concepts that appear within the views of a model.

5) Model. A *model* is a packaging of one or more views along with the narrative descriptions and specification language for the view and view components (classes, responsibilities, etc.) called out in the model's views.

4.2 IDEF1X notation

The IDEF1X notation includes diagrams, free text, and the specification language.

a) Diagrams present the subject domains, classes, responsibilities, relationships, attributes, operations, and constraints of interest in a view.

b) Free text is used for labels, statements of responsibilities, narrative description, and notes.

[9] Every view automatically contains the classes, relationships, etc., of the $IDEF1X_{97}$ language metamodel, although these classes, relationships, etc., are customarily not shown in the graphics and the modeler does not declare them.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

c) The specification language is used for names, requests, statements of pre- and post-conditions, and the realization of operations, constraints, and derived attributes.

4.2.1 Example IDEF1X diagram

Figure 3 illustrates some of the aspects of classes and relationships that are described in Clause 5. The classes `hotel` and `room` are state classes; `room` is a dependent state class, and `hotel` is an independent state class (see 5.2). `temperature` is a value class (see 5.3). The relationship between `hotel` and `room` says that *"each hotel contains rooms"* (see 5.5). While not shown in Figure 3, Clause 5 also discusses the generalization of classes in 5.4.

Responsibility names, if shown, are listed inside the rectangles in the graphic diagram. Responsibilities are described in Clause 6. This diagram does not distinguish graphically among attribute, participant, and operation properties. The annotations to do so are covered in 6.3.

Specifying the type of an attribute is optional. In Figure 3, types are shown only for the `temperature` class, i.e., `fahrenheit`, `celsius`, `kelvin`, and `rep` are all of type `real`. Within the `temperature` class, the attribute `rep` is marked "private" as indicated by the double bar preceding the attribute name; this is the hidden value (which could be any one of the other values or perhaps something different entirely) that the `temperature` class uses to represent temperature values.[10] The property `guest` is marked with an `(o)` to designate that a value is optional; that is, a room may be vacant. The full set of suffix annotations are covered in 6.3.

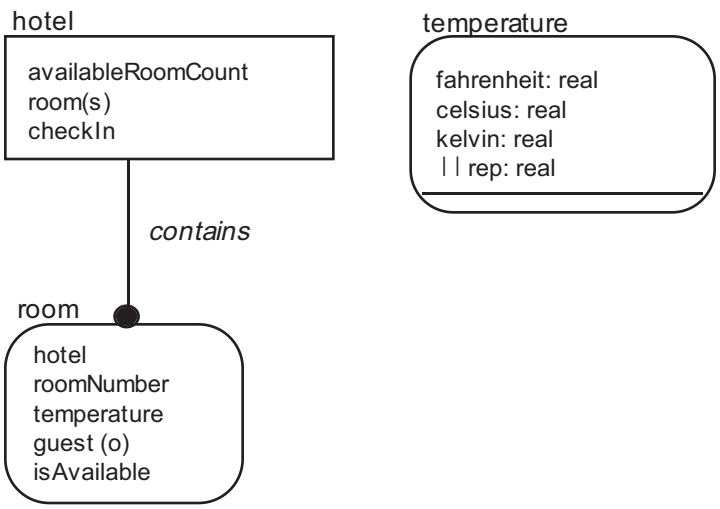

Figure 3—An example IDEF1X diagram

In addition to attributes, the property list displayed for a class may include participant properties and operations. Because of the relationship between `hotel` and `room`, the class `room` has a participant property named `hotel,` and the class `hotel` has a participant property named `room(s)`, where the `(s)` suffix reflects its plural (collection-valued) nature. While Figure 3 does display these participant properties, they are not always displayed in a diagram, as explained in 6.5. This figure also illustrates an operation, `checkIn`, for the class `hotel`.

It is sometimes useful to illustrate instance examples for classes. Two methods for doing so are shown in Figures 4 and 5. Figure 4 portrays what is referred to as a *sample instance diagram*. In this figure, sample instances of `hotel`, `room`, and `temperature` are shown as separate shapes. Above each rectangle is

[10]See 6.4.5 for a description of how representation properties are used to derive the public attribute values.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

written the intrinsic object identifier (oid) of the instance. Inside the rectangles are written property names and their values. Collection-valued property instance values are shown within brackets with a literal form appropriate to the collection class (here with curly braces for `set`).

Figure 5 shows what is referred to as a *sample instance table*. The figure shows the same sample instances of `hotel`, `room`, and `temperature`, this time in the form of tables representing the classes. Columns in each table represent properties and the cells hold property values. The oid of each instance is shown to the left of the row representing the instance. When a property has no value, a "double dash" (`--`) is shown in sample instance tables and diagrams.

In the diagrams and specification language, proper nouns denoting a specific but (at the time of writing) unknown instance of a class are written with an initial capital letter. For example, `TheHotel` and `R1` each achieve the same end—standing (arbitrarily) for instances of `hotel` and `room`, respectively. Any other nouns are written with an initial lowercase letter or surrounded by single quotes. Sample instance diagrams and tables are discussed more fully in 5.2 and 5.3.

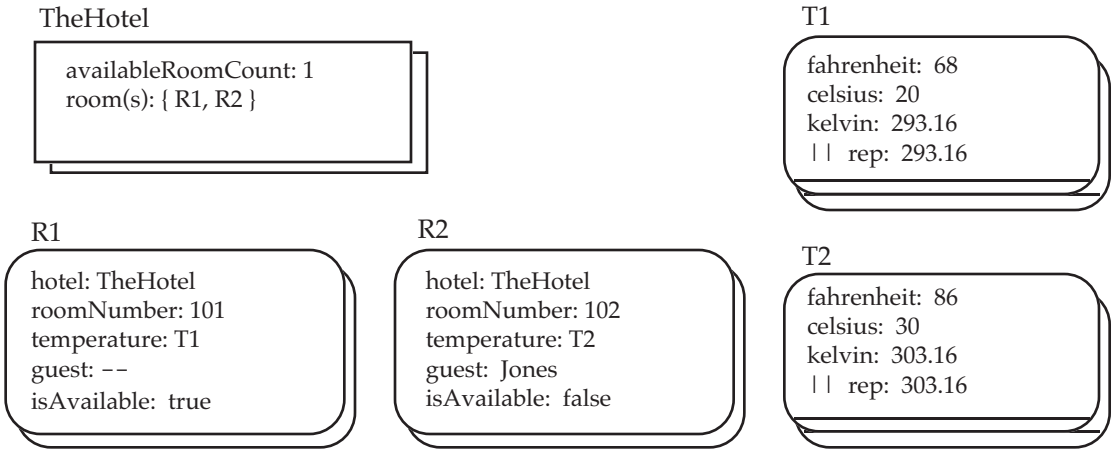

Figure 4—Sample instance diagram

4.2.2 Example specification language

IDEF1X includes a declarative specification language called the Rule and Constraint Language (RCL). Below are four examples of its use.[11] The specification language is fully described in Clause 7.[12]

4.2.2.1 Example attribute derivation specification

The specification language is used to express the realization of operations, constraints, and derived attributes and participant properties. For example, the `availableRoomCount`[13] derived attribute of `hotel` in Figure 3 has the following realization:

[11]The first and third examples also illustrate that the specific (but unknown) instance in a declaration may be named in various ways, e.g., as `TheHotel` or as `Self`. This standard does not proscribe any particular style. For example, while `Self` is a form familiar to a Smalltalk user, a C++ or Java user might have selected `This`.

[12]Several equivalent forms for RCL are described in Clause 7. The examples throughout do not attempt to show all equivalent forms.

[13]Several forms of expressing a name are allowed by the lexical rules specified in 4.2.3. This form is used for consistency throughout.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

hotel

	availableRoomCount	room(s)
TheHotel	1	{R1, R2}

room

	hotel	roomNumber	temperature	guest	isAvailable
R1	TheHotel	101	T1	--	true
R2	TheHotel	102	T2	Jones	false

temperature

	fahrenheit	celsius	kelvin	|| rep
T1	68	20	293.1	293.16
T2	86	30	303.1	303.16

Figure 5—Sample instance tables

```
hotel: TheHotel has availableRoomCount: N if_def
             AvailableRooms is [ Room where
                Room is TheHotel..room(s)..member,
                Room has isAvailable ],
             N is AvailableRooms..count.
```

`TheHotel` is the instance of `hotel` that is the receiver of the request for the `availableRoomCount` derived attribute value. The specification language says that in order for `TheHotel` to have an `availableRoomCount` value of `N`, it is necessary and sufficient that

a) `AvailableRooms` is a list containing every `Room` where the `Room` is `TheHotel`'s `room`, and the `Room` `isAvailable`, and
b) `N` is the `AvailableRooms`' `count`.

The specification of derived attributes and participant properties is discussed more fully in 6.4 and 6.5, respectively.

4.2.2.2 Example operation realization specification

Another example provides a declarative specification of the operation `checkIn` in Figure 3.

```
hotel: Self has checkIn: Guest if_def
             Self has room(s)..member: R,
             R has isAvailable,
             R has guest:= Guest,
             R has isAvailable:= false.
```

When requested, a particular hotel instance will "check in" the identified guest if the hotel has an available room. A successful execution of the `checkIn` operation assigns the guest to the available room and makes that room unavailable. The specification of operations is discussed more fully in 6.6.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

4.2.2.3 Example constraint specification

The specification language can be used to state constraints—statements of facts that are required to be true in order that the model conform to the world being modeled. An example of the declarative specification of the constraint `hotelOwnsTv` (see Figure 70) is

```
tvInARoom: Self has hotelOwnsTv if_def
             Self..room..hotel == Self..tv..hotel.
```

This constraint states that a television in a room has "valid ownership" if the hotel that contains the room of this television and the hotel that owns the television are precisely the same hotel. The specification of constraints is discussed more fully in 6.7.

4.2.2.4 Example query specification

The specification language is also used to make queries (requests for property values). In Figure 3, the value class `temperature` represents abstract temperatures. Each instance of `temperature` has both a `fahrenheit` and a `celsius` value (as well as a `kelvin` value). The specification language sentence

```
T is temperature with fahrenheit: 68,
C is T..celsius.
```

is a query that identifies `T` as the instance of `temperature` with a `fahrenheit` value of `68` and requests that instance's corresponding `celsius` value. `C` is whatever the `celsius` value is for that same instance `T`. If the sentence is executed, `C` will be solved for and found to be `20`.

4.2.3 Lexical rules

A *name* is a word or phrase that designates some model construct (such as a class, responsibility, subject domain, etc.). A *label* is a word or phrase that is attached to or part of a model graphic; it typically consists of a model construct's name (or one of the aliases) and may contain additional textual annotations (such as a note identifier). Refer to Clause 7 for the formal RCL syntax.

Free text shall be used for the names and labels of IDEF1X constructs, according to the following rules:

4.2.3.1 Naming

a) An *unquoted name* (i.e., a name not surrounded by single quotes) shall contain only alphanumeric characters and underscores.
b) A *quoted name* (i.e., a name surrounded by single quotes) may contain any character.
c) A quoted name shall specify an imbedded single quote by two adjacent single quotes.
d) A *simple name* shall be an unquoted name or a quoted name.
e) A *qualified name* shall be a series of simple names separated by colons.
f) The single quotes surrounding a quoted name may be omitted in the model graphics.
g) A name of the form `#K`, where `K` is a constant, shall be used to denote a state class instance that is known at the time of writing (see Figure C.23)
h) A name denoting a specific but (at the time of writing) unknown instance of a class shall begin with an uppercase letter.
i) Any other name shall begin with an initial lowercase letter or shall be surrounded by single quotes.
j) A name may not exceed 254 characters in length. All characters shall be treated as significant.
k) If a valid unquoted name begins with a lowercase letter, the name shall be considered equivalent to the same name surrounded by single quotes.
l) If a name used in model graphics is not a valid unquoted name, then it shall be surrounded by single quotes when used in RCL.
m) Each keyword shall be in lowercase.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

4.2.3.2 Label

a) Except for the first character, a label shall be case insensitive, i.e., "`A`" and "`a`" are equivalent.
b) A label may "wrap" and be displayed as multiple lines.
c) A label displayed outside its associated graphic box (e.g., a class label outside its class box) may not extend beyond the right bounding line of the graphic box.
d) A label displayed inside its associated graphic box (e.g., a class label inside its class box or a displayed property signature) may not extend beyond the bounding lines of the graphic box.
e) On a multiline label, any annotation symbols (e.g., note numbers) shall be included at the end of the last line of the label.
f) All *whitespace* (spaces, tabs, etc.) in a label shall be preserved.

An example of a state class label that includes a note is

```
purchase-Order-Item (12)
```

An example of a multi-line relationship verb phrase label that includes a note is

```
is
assigned
to (5)
```

5. Class

People mentally classify things that are similar in some sense into a *class* named for that sense and representing all such similar things. Everyone does this classification; it is part of common sense. The things that are classified in this way are individual things, distinct from all other things. A class is an abstraction of the knowledge and behavior of a set of similar things.

5.1 Introduction

There are two kinds of class: state class and value class.[14] The distinction is introduced in 5.1.3 and more fully discussed in 5.2 and 5.3, respectively. This subclause describes the concepts that apply to the concept of *class* in general.

Anything that is categorized into a class is said to be an *instance* of the class. An instance possesses knowledge, exhibits behavior, and obeys rules. These notions are collectively referred to as the instance's *responsibilities*. A class abstracts the responsibilities in common to its instances. Initially, a responsibility may simply be stated in general terms and not distinguished explicitly as an attribute, participant property, operation, or constraint. Also, aggregate responsibilities may be identified, rather than individual properties. Broadly stated responsibilities are eventually refined as specific properties and constraints. In addition to these instance-level responsibilities, a class may also have *class-level* responsibilities in the form of attributes, operations, and constraints. These class-level responsibilities constitute the knowledge, behavior, and rules of the class as a whole. Responsibilities are described in detail in Clause 6.

5.1.1 Class semantics

5.1.1.1 Identity

Each instance is considered to have a unique, intrinsic *identity* that is independent of its property values or the classes to which it belongs. It is an instance's unique identity that distinguishes it from all other instances.

[14]This distinction is explicitly made in "The Evolution of Domains" [B4] and ODMG-93 [B11] where state classes and value classes are referred to mutable and immutable classes, respectively. However, this distinction is often left implicit in object model formulations.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

The notion of identity is first of all a concept. Some concrete representation for the concept must be used to show examples of instances with identity, to formalize the notion of identity, or to support the notion in programming languages or database systems. This concrete representation is referred to as the intrinsic oid of the instance.

5.1.1.2 Intension/extension

The notion of class has an intensional and an extensional aspect. The *intension* reflects the sense of the class. The *intensional set* is determined by the meaning of the class. All possible things that are similar in the sense of the class are members of the intensional set.

The *extensional set* of instances contains the currently existing instances. The extensional set is always a subset of the intensional set.[15] The instances in the extensional set correspond to the database and data modeling notion of *instance*. The extensional set is sometimes called the *current extent*. Unless otherwise noted, *instance* means an existing instance, that is, a member of the current extent.

5.1.1.3 State class/value class

A *state class* is one in which the extensional set of instances is a time-varying subset of the intensional set of instances. An instance changes state when it is born, when it takes on attribute or participant property values, when it changes those values, or when it dies. A class of such instances is called a state class. The class `registeredVoter` is an example of a state class. The extensional set of registered voters varies over time. `Chris Jones` could be an instance of the state class `registeredVoter` at any particular time.

Instances that do not change state are pure values. A class of such instances is called a *value class*. A value class is one in which the extensional set of instances is fixed and equal to the intensional set of instances. The instances act as pure values, like an integer or a mathematical set. It makes no sense to have duplicate instances—there is only one `17`, only one `0`, only one empty set, and so on. Because the instances of a value class act like values, a value class instance is sometimes called a *value*, but it is still an instance.

Value class instances cannot be created, updated, or deleted. It makes no sense to update `17`; it would not be `17` any more. If a mathematical set has a member removed or a new one is added, it is not the same set any more. Everyday examples of value classes include `date` and `time`. Again, "updating" a date or a time makes no sense; it would yield a different date or time, not the same one changed in some way.

Everything that is said about classes, instances, and properties applies to both state and value classes unless specifically restricted. For example, value classes do not participate in relationships. Also, the representation of identity is typically different for state and value classes—for state classes, the identity is represented by the intrinsic identifier; for value classes, the identity is represented by the value. The details of state classes and value classes are discussed in detail in 5.2 and 5.3, respectively.

5.1.1.4 Abstract data type

A class can be considered an abstract data type. Traditionally, an *abstract data type* (ADT) is a data type for which the user of the data type can create instances of the data type and operate on those instances, but for which the range of valid operations available to the user does not depend in any way on the internal representation of the instances or the way in which the operations are realized. The data is abstract in the sense that values in the extent, i.e., the concrete values that represent the instances, are

— Any set of values that support the operations, and
— Irrelevant to the user.

[15]This fact forms the foundation of *information*. Otherwise, everything that could possibly be true is in the database rather than just those things that are presently true.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

An ADT identifies the operations on the data as part of the specification of the data and separates what can be done (the interface) from how it is done (the realization).[16]

For example,[17] the interface declaration of the value class `temperature` specifies

— A way to denote a unique temperature by its Fahrenheit or Celsius or Kelvin value, and
— Operations to obtain the Fahrenheit, Celsius and Kelvin values.

The realization of `temperature` specifies

— Whether the temperature is to be represented by its Fahrenheit or Celsius or Kelvin value, and
— The appropriate rules for each operation, depending on the representation choice.

The nature of the abstract values in the extent of the `temperature` value class is exemplified by the sentence "32 Fahrenheit is the same thing as 0 Celsius." That thing is the instance of temperature. When the `fahrenheit` operation is applied to the instance, it yields `32`. When the `celsius` operation is applied to the same instance, it yields `0`.

For another example, the interface declaration of the value class `vector` specifies

— A way to denote a unique vector by its coordinate values or by its magnitude and direction,
— Operations to obtain the values of the coordinates, magnitude, and direction, and
— Operations such as adding a vector to a vector.

The realization of the value class `vector` specifies

— Whether the vector is to be represented by coordinate values, or by magnitude and direction values, and
— The rules for each operation.

5.1.1.5 Built-in class/user-defined class

A few classes, such as `object`, `class`, `integer`, `real`, `string`, and `list`, are assumed to be *built-in* to the IDEF1X language. These classes provide properties such as instance creation, instance deletion, access to the instances of a class, and numeric, string, and list operations. Classes that are not built-in are *user-defined*. The built-in state classes and built-in value classes are presented in Clause 10 and Annex D.

5.1.1.6 Collection class

The typical class represents instances that are atomic. Some classes, however, have instances that are themselves collections of other instances. Such a class is called a collection class. A *collection class* is a kind of class in which each instance is a group of instances of other classes. Examples of collection classes are `list`, `set`, and `bag`.

5.1.1.7 Parameterized collection class

The built-in `list`, `set`, and `bag` collection classes are untyped, i.e., their instances can be collections of anything. However, a collection class can be restricted to hold only instances of a specified type (class). This kind of collection class is called a *parameterized collection class*, a class in which a parameter specifies the class of the instances that the collection may contain.

[16] "Interface" and "realization" are explained in 6.1. See also [B12].
[17] These examples are developed further in 5.3, 6.4, and 6.6.

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

The built-in collection class generators `list ( T )`, `set ( T )`, and `bag ( T )` permit typed collection classes to be specified. A class is "generated" in the sense that the classes `set ( real )` and `set ( integer )` are two distinct classes, not a single class with a parameter variable. For example, the instances of `set ( real )` are restricted to be sets of real numbers. The collection class `set ( integer )` is a different collection class; its instances are restricted to be sets of integers.

The parameter, `T`, can be any built-in or user-defined class. All the properties and constraints of the built-in collection classes apply to the generated collection classes. (See Clause 7 for a further discussion of collection classes.)

5.1.1.8 Parameterized pair class

The built-in `pair` class is untyped, i.e., its instances can be pairs of anything. However, a pair class can be restricted to hold only instances of specified types (classes). This kind of pair class is called a *parameterized pair class*, a class in which two parameters specify the classes of the instances that the pair may contain.

The built-in pair class generator `pair ( T1, T2 )` permits typed pair classes to be specified. A class is "generated" in the sense that the classes `pair ( real, integer )` and `pair ( integer, string )` are two distinct classes, not a single class with parameter variables.

The parameters, `T1` and `T2`, can be any built-in or user-defined classes. All the properties and constraints of the built-in pair class apply to the generated pair classes. (See Clause 7 for a further discussion of pair classes.)

5.1.2 Class syntax

5.1.2.1 Graphic

a) A class shall be represented as a rectangle of the shape appropriate to its class.

The shapes for state class are specified in 5.2.2.1. The shape for value class is specified in 5.3.2.1.

5.1.2.2 Label

a) Each class displayed in a view shall be assigned a label.
b) The label of a class in a view shall consist of the class name or one of its aliases.

The syntax for state class labeling is specified in 5.2.2.2. The syntax for value class labeling is specified in 5.3.2.2.

5.1.2.3 Sample instances

a) For every model it shall be possible to present sample instances that validate the model.
b) When provided, sample instances shall be presented in one of two forms, as a *sample instance diagram* or *sample instance table*.

The representation of sample instances of a state class is specified in 5.2.2.3 through 5.2.2.6. The representation of sample instances of a value class is specified in 5.3.2.3 through 5.3.2.6.

5.1.3 Class rules

5.1.3.1 Naming

a) A class shall have both a simple (unqualified) name and a fully qualified name.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012

b) The simple name of a class shall be a noun or noun phrase.

c) A class shall be given a simple name as one would refer to a single instance of the class. Typically that name is singular in form, not plural.

For example, an instance of a state class may represent a set of things. If a state class instance represents a collection of things, as in a set of playing cards, a plural noun (e.g., `cards`) could be used for the state class name (although a singular form such as `deck` would also be appropriate).

For example, an instance of a value class may have a list as its internal representation. If a value class instance is plural, as in a set of coordinates, a plural noun (e.g., `coordinates`) would be appropriate for the value class name.

d) A class shall have a fully qualified name,[18] as follows:

1) The fully qualified name of a class with a simple name `Csn` in a view named `Vn` shall be `Vn:Csn`.

2) The fully qualified name of a class with the simple name `Csn` in a view with no parent view shall be just `Csn`.

5.1.3.2 Responsibilities

a) A class may have any number of responsibilities.

5.2 State class

A state class[19] represents instances with changeable state. Its instances can come and go, and can change state over time, i.e., their property values can change.

A state class is a class that represents a set of real or abstract objects (people, places, events, ideas, things, combinations of things, etc.) that have common knowledge and behavior and adhere to common constraints. An individual member of the set is referred to as a *state class instance* (simply, *instance*). A real world thing may be represented by more than one state class. For example, `John Doe` can be an instance of both the state class `employee` and the state class `buyer`. Furthermore, an instance may represent a concept involving a combination of real world things. For example, `John` and `Mary` could be the participants in an instance of the state class `marriedCouple`.

5.2.1 State class semantics

5.2.1.1 Instance identity

For a state class, an oid represents the concept of identity. In terms of a representation system (i.e., the examples, formalization, or software), the oid stands for the instance. In a sample instance diagram or sample instance table of state class instances, each row has an associated oid. For example, oids are shown in the sample instance diagram in Figure 13 and the sample instance tables in Figure 14. Note that an oid is not an attribute; the oid is always hidden from the client.

5.2.1.2 Independent state class/dependent state class

A state class instance is distinguished from all other instances of its class because of its intrinsic identity; that is, the typical state class is considered to be an *independent state class*. Its instances have existence, knowledge, and behavior independent of other instances. However, there may be cases where it makes no sense to have a class instance by itself and unrelated to an instance of another class(es) and, furthermore, where it makes no sense to change the instance(s) to which it relates. This type of class is referred to as a

[18] See also 8.1.3.1 and 10.7.2.

[19] "State class" is not to be confused with the notion of a "state machine."

Documents
Document Sharing Hub

Copyright © 1999 IEEE. All rights reserved.

Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.

Full document details: ISO/IEC/IEEE 31320-2-2012

dependent state class. A dependent state class instance is by its very nature *intrinsically* related to certain other state class instance(s).[20]

Both cases are illustrated in Figure 6. In this example, `room` is a dependent state class, dependent on the independent state class `hotel`. A room of the hotel cannot exist without its hotel. It makes no sense to separate the room from the hotel; the room would not exist. Furthermore, a hotel room is intrinsically a part of some one specific hotel (at least in the example), and it makes no sense to change it to a different hotel—to do so would yield a different hotel room, not the same one changed.

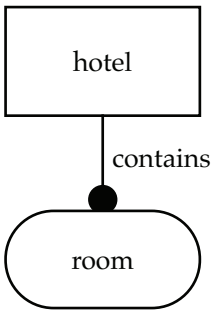

Figure 6—Independent and dependent state classes

State class dependency can be expressed more precisely. A state class `d` is dependent on another state class `c` if and only if every instance `D` of `d` is related to exactly one instance of `c`, and `D` cannot be updated to be related to any other instance of `c` or to no instance.

The distinction between "independent" and "dependent" has historically proven useful in IDEF1X modeling. In building a model, the independent state classes usually emerge first. In seeking to understand a model, a familiarity with the independent classes is required before the meaning of the dependent classes can be understood.

5.2.2 State class syntax

5.2.2.1 Graphic

a) A state class shall be represented as a rectangle.
b) An independent state class rectangle shall have square corners, as illustrated in Figure 7.
c) A dependent state class rectangle shall have rounded corners, as illustrated in Figure 7.

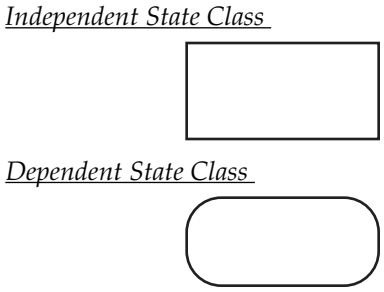

Figure 7—State class syntax

[20]Although an instance of a state class may depend on another instance, it still has its own identity, as indicated previously.

Copyright © 1999 IEEE. All rights reserved.
Authorized licensed use limited to: Tsinghua University. Downloaded on March 07,2026 at 00:42:45 UTC from IEEE Xplore. Restrictions apply.
Documents
Document Sharing Hub
Full document details: ISO/IEC/IEEE 31320-2-2012