

**Information technology —
Portable Operating System Interface
(POSIX™)
system administration —
Part 2:
Software administration**

**Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved.**

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE 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 the 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, the IEEE and the members of its technical 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 Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331

IEEE Standards documents may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user of such technology to obtain all necessary permissions.

Contents

	PAGE
Section 1: General	1
1.1 Scope	1
1.2 Normative References	3
1.3 Conformance	3
1.4 Test Methods	8
Section 2: Terminology and General Requirements	9
2.1 Conventions	9
2.2 Definitions	12
2.3 Dependencies on Other Standards	27
Section 3: Software Structures	29
3.1 Software_Collection	30
3.2 Distribution	32
3.3 Media	33
3.4 Installed_Software	33
3.5 Vendor	34
3.6 Software	35
3.7 Products	36
3.8 Bundles	39
3.9 Filesets	42
3.10 Subproducts	44
3.11 Software_Files	45
3.12 Files	46
3.13 Control_Files	48
Section 4: Software Administration Utilities	51
4.1 Common Definitions for Utilities	51
4.2 swask — Ask for user responses	90
4.3 swconfig — Configure software	93
4.4 swcopy — Copy distribution	97
4.5 swinstall — Install software	102
4.6 swlist — List software catalog	114
4.7 swmodify — Modify software catalog	118
4.8 swpackage — Package distribution	121
4.9 swremove — Remove software	125
4.10 swverify — Verify software	131
Section 5: Software Packaging Layout	137
5.1 Directory Structure	137
5.2 Software Definition File Format	141
5.3 Serial Format and Multiple Media	158
Annex A (informative) Bibliography	161

Annex B (informative) Rationale and Notes	165
B.1 General	165
B.2 Terminology and General Requirements	167
B.3 Software Structures	172
B.4 Software Administration Utilities	188
B.5 Software Packaging Layout	234
Annex C (informative) Sample Files	243
C.1 Defaults File	243
C.2 Product Specification File	245
C.3 Software Packaging Layout	247
C.4 INDEX File	248
C.5 INFO File	249
C.6 Control Script	250
Annex D (informative) Portability Considerations	251
D.1 User Requirements	251
D.2 Portability Capabilities	253
D.3 Profiling Considerations	254
Alphabetic Topical Index	257

FIGURES

Figure 5-1 – Example of Software Packaging Layout	138
Figure B-1 – Roles in Software Administration	168
Figure B-2 – Example of Software Structure	172
Figure B-3 – Software Object Containment	175
Figure B-4 – Software Object Inheritance	176
Figure B-5 – Fileset State Transitions (Within Distributions)	208
Figure B-6 – Fileset State Transitions (Within Installed Software)	209
Figure B-7 – Installation State Changes	216
Figure B-8 – Order of Install Operations	220
Figure B-9 – Order of Remove Operations	231
Figure B-10 – SDU INFO file format	237
Figure B-11 – SVR4 pkgmap file format	237

TABLES

Table 2-1 – Typographical Conventions	9
Table 3-1 – Attributes of the Software_Collection Common Class	31
Table 3-2 – Attributes of the Distribution Class	32
Table 3-3 – Attributes of the Media Class	33
Table 3-4 – Attributes of the Installed Software Class	34
Table 3-5 – Attributes of the Vendor Class	34
Table 3-6 – Attributes of the Software Common Class	35
Table 3-7 – Attributes of the Product Class	37

Table 3-8	– Attributes of the Bundle Class	40
Table 3-9	– Attributes of the Fileset Class	42
Table 3-10	– Attributes of the Subproduct Class	44
Table 3-11	– Attributes of the Software_Files Common Class	45
Table 3-12	– Attributes of the File Class	47
Table 3-13	– Attributes of the Control File Class	49
Table 4-1	– Software_spec Version Identifiers	56
Table 4-2	– Script Return Codes	73
Table 4-3	– Event Status	74
Table 4-4	– General Error Events	75
Table 4-5	– Session Events	76
Table 4-6	– Analysis Phase Events	78
Table 4-7	– Execution Phase Events	83
Table 4-8	– Return Codes	89
Table 4-9	– Default Levels	116
Table 5-1	– File Attributes for INFO File	153
Table B-1	– Possible Attributes of a Host Class	177
Table B-2	– Mapping from Software to DMTF Component ID	180
Table B-3	– Comparison of Some Existing Practices	192
Table B-4	– Comparison of Software Administration Packages	217
Table B-5	– Comparison of Existing Practice for Software Packaging	228
Table B-6	– Packaging Layout Comparisons	236
Table D-1	– 1387.2 Portability Capability Summary	253

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) together form a system for worldwide standardization as a whole. 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 nongovernmental, 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. Draft International Standards adopted by the joint technical committee are circulated to national bodies for approval before their acceptance as International Standards. They are approved in accordance with procedures requiring at least 75% approval by the national bodies voting.

International Standard ISO/IEC 15068-2: 1997 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

ISO/IEC 15068 consists of the following parts, under the general title *Information technology—Portable operating system interface (POSIX)*:

- *Part 1: (reserved for future use)*
- *Part 2: Software administration*
- *Part 3: User and group account administration* (in DIS Ballot)
- *Part 4: Print administration* (under development)

All of the annexes found in this part of ISO/IEC 15068 are for information only.

Introduction

(This Introduction is not a normative part of ISO/IEC 15068-2 Information technology — Portable Operating System Interface (POSIX) system administration — Part 2: Software administration, but is included for information only.)

System administration utilities vary widely between vendors, being an area where there are currently no formal standards that have proved to be significant in practice. This makes the task of system administration difficult. The objective of this part of ISO/IEC 15068 is to address this problem for software administration, a specific area of system administration, and to contribute to the overall solution of administering computing environments, both stand-alone and distributed.

In pursuit of this goal, this part of ISO/IEC 15068 defines a software packaging layout, a set of information maintained about software, and a set of utility programs to manipulate that software and information. These definitions provide the flexibility necessary for system administrators to enforce policies suitable to their environments.

Organization of the Standard

The standard is divided into the following sections:

- (1) General
- (2) Terminology and General Requirements
- (3) Software Structures
- (4) Software Administration Utilities
- (5) Software Packaging Layout

Also included are the following annexes:

- Bibliography (Annex A)
- Rationale and Notes (Annex B)
- Sample Files (Annex C)
- Portability Considerations (Annex D)

This introduction and the annexes are not considered a normative part of the standard.

Conformance Measurement

In publishing this part of ISO/IEC 15068, both IEEE and the POSIX.7.2 developers simply intend to provide a yardstick against which various operating system implementations may be measured for conformance. It is not the intent of either the IEEE or POSIX.7.2 developers to measure or rate any products or to reward or sanction the product of any vendor as standard by these or any other means. The responsibility for determining the degree of conformance or lack thereof with this part of ISO/IEC 15068 rests solely with the individual evaluating the product claiming to be in conformance with the standard.

Base Documents

Much of the original text came to the developers of this part of ISO/IEC 15068 from UNIX System Laboratories (the **pkg*** utilities) and Hewlett-Packard (HP Software Distribution Utilities). For further details and comparisons of various existing practices, see B.4 and B.4.5.

Extensions and Supplements to This Standard

Activities to extend this standard to address additional requirements are in progress and similar efforts can be anticipated in the future. This is an outline of how these extensions will be incorporated and how users of this document can keep track of that status.

Extensions are approved as supplements to this document following the IEEE standards procedures and eventually as International Organization for Standardization/International Electrotechnical Committee (ISO/IEC) standards.

Approved supplements are published separately and distributed with orders from the IEEE for this document until the full document is reprinted and such supplements are incorporated in their proper positions.

If there are any questions about the completeness of your version, you may contact the IEEE Computer Society, (202) 371-0101, or the IEEE Standards Office, (908) 562-3800, to determine what supplements have been published.

Supplements may contain either required functions or optional facilities. Supplements may add additional conformance requirements (see 1.3, which defines new classes of conforming systems or applications).

It is undesirable (but perhaps unavoidable) for supplements to change the functionality of the already defined facilities.

If you are interested in participating in addressing IEEE 1387 issues and developing the IEEE P1387 standards, please send your name, address, and phone number to the Secretary, IEEE Standards Board, P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08844-1331, USA.

When writing, ask to have your letter forwarded to the chairperson or appropriate reviewer/developer of IEEE Std 1387.2-1995.

ISO/IEC 15068-2 was prepared by the IEEE P1387.2 Working Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society. At the time this part of ISO/IEC 15068 was approved, the membership of the IEEE P1387.2 Working Group was as follows:

Portable Applications Standards Committee

Chair:	Lowell Johnson
Vice-Chair:	Charles Severance
Functional Chairs:	Andy Bihain
	Jon Spencer
	Andrew Josey
	Jay Ashford
	Barry Needham
Treasurer:	Peter Smith
Secretary:	Charles Severance

IEEE P1387 Working Group Officials

Chair:	Martin Kirk
	Steve Carter (1988-1990)
Vice-Chair:	Jay Ashford
	David Hinnant (1988-1991)
Technical Editor:	Matt Wicks (1991-1993)
	Robert Robillard (1991-1993)
	Shoshana O'Brien (1989-1991)
	Bob Baumann (1988-1990)
Secretary:	Bernard Kinsler (1990-1995)
	John Pokladnik (1990)
	Mark Colburn (1989)

IEEE P1387.2 Working Group Officials

Chair:	Jay Ashford
Technical Editors:	Jay Ashford, George Williams
	Matt Wicks (1991-1993)

IEEE P1387.2 Technical Reviewers

Jay Ashford	Jerry Rubin	George Williams
	Matt Wicks	

IEEE P1387.2 Working Group

Richard Alexander	Louis Imershein	Ken Nicholson
Barrie Archer	John Jobs	Per Pedersen
Jay Ashford	Jim Johnson	Daryl Roberts
Shane Claussen	Judy Kale	Helmut Roth
Jim Darling	Martin Kirk	Jerry Rubin
Frances Dodson	Esti Koen	Nigel Titley
Frank Dogil	Steve Lamotte	Stephe Walli
Janet Frazer	Sean Landis	Matthew Wicks
Jay Goldberg	Fu-Tin Man	George Williams
Michael Gutmann	Norbert Marrek	Neil Winton
Steve Howell	Laura Micks	Jane Zysk
David Humphreys	Tom Murphy	

The following persons provided valuable input during the balloting period:

Francesco Borgna	Shane P. McCarron	Mike Ryan
Theodore Collins	Brenda Parsons	Larry Spieler
Cheng Hu	Dieter Putatzki	Marc J. Stephenson
	Walter Wong	

The following persons were on the balloting committee:

Barrie Archer	Geoff Hall	John S. Morris
Jay Ashford	Barry Hedquist	Mo Oloumi
Jason Behm	Joseph Hungate	Paul Rabin
Michael E. Browne	Louis Imershein	David Radford
Dana Carson	Hal Jespersen	Rick Roelling
Shane Claussen	Judy S. Kerner	Frank Rone
Frances Dodson	Lawrence Kilgallen	Jerrold Rubin
Ron Elliott	Martin J. Kirk	James G. Tanner
Michael E. Falck	Esti Koen	Mark-Rene Uchida
David Fiander	George Kriger	Matthew Wicks
Dan Geer	Thomas M. Kurihara	George Williams
Michel Glen	Sean Landis	Walter Wong
Dave Grindeland	Jim Moore	Oren Yuen

When the IEEE Standards Board approved this standard on June 14, 1995, it had the following membership:

E. G. "Al" Kiener, *Chair*

Donald C. Loughry, *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril	Richard J. Holleman	Marco W. Migliaro
Clyde R. Camp	Jim Isaak	Mary Lou Padgett
Joseph A. Cannatelli	Ben C. Johnson	John W. Pope
Stephen L. Diamond	Sonny Kasturi	Arthur K. Reilly
Harold E. Epstein	Lorraine C. Kevra	Gary S. Robinson
Donald C. Fleckenstein	Ivor N. Knight	Ingo Rusch
Jay Forster*	Joseph L. Koepfinger*	Chee Kiow Tan
Donald N. Heirman	D. N. "Jim" Logothetis	Leonard L. Tripp
	L. Bruce McClung	

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
Richard B. Engelman
Robert E. Hebner
Chester C. Taylor

Rochelle L. Stern
IEEE Standards Project Editor

Information technology — Portable Operating System Interface (POSIX) system administration — Part 2: Software administration

Section 1: General

1.1 Scope

This part of ISO/IEC 15068 defines a software packaging layout and utilities that operate on that packaging layout as well as software installed from that packaging layout. The scope of this part of ISO/IEC 15068 is administration of software across distributed systems. This administration includes, but is not limited to, packaging of software for distribution, distribution of software to systems, installation and configuration of software on systems, and removal of software from systems.

This part of ISO/IEC 15068 is motivated by many factors, including a desire by system administrators and software suppliers to have a common way of installing and removing software. To meet the needs of these groups, this part of ISO/IEC 15068 consists of several components, listed below. The readers of this part of ISO/IEC 15068 include system administrators, suppliers of software that implement this part of ISO/IEC 15068, and suppliers of software that use implementations of this part of ISO/IEC 15068. Readers in each of these categories may find their attention drawn to different sections.

The key components are listed below.

Software structures

This part of ISO/IEC 15068 defines a hierarchical set of structures used to define software. Information is kept about the software based on these structure definitions. The structure definitions apply both to installed software and to software prepared for installation

1.1 Scope

22 but not yet installed.

23 **Software packaging layout**

24 This part of ISO/IEC 15068 defines the organization of software on a
25 distribution medium, the information held about that software, and
26 the way in which such information is represented. This enables both
27 portability of software distributions across systems of different archi-
28 tecture, and the use of different media to distribute software (includ-
29 ing both file system and serial image forms).

30 **Information kept about software**

31 This part of ISO/IEC 15068 defines the information that is held about
32 software, both installed software and distributions. This provides a
33 consistent view of software, even when that software is provided from
34 various sources. The way in which the information is held is
35 undefined within this part of ISO/IEC 15068.

36 **Utilities to administer software**

37 This part of ISO/IEC 15068 defines a utility to convert software into
38 the packaging layout, known as a distribution. This part of ISO/IEC
39 15068 also contains utilities to examine the information in a distribu-
40 tion, copy software from one distribution to another, install software
41 from a distribution, remove software from a distribution, and verify
42 the integrity of a distribution. There are also utilities for configuring
43 installed software, verifying the integrity of installed software, exa-
44 mining and modifying the information held about installed software,
45 and for removing installed software from a system. This provides
46 administrators with a consistent method of dealing with software
47 across all conforming systems.

48 **Distributed software administration**

49 This part of ISO/IEC 15068 defines the concepts, and the utility syn-
50 tax and behaviors, for managing software in a distributed environ-
51 ment. This includes the concept of different software administration
52 roles (developer, packager, manager, source, target, and client).
53 Different utilities involve different roles, and different roles may be
54 distributed across multiple systems within a single command execu-
55 tion.

56 This part of ISO/IEC 15068 is based upon the knowledge of, and documentation
57 for, existing programs that assume an interface and architecture similar to that
58 described by POSIX.1 {2}¹⁾ and POSIX.2 {3}. Any questions regarding the definition
59 of terms or the semantics of an underlying concept should be referred to
60 POSIX.1 {2} and POSIX.2 {3}. This part of ISO/IEC 15068 does not require the use
61 of any specific programming language and, in particular, does not require the use
62 of the C language. This part of ISO/IEC 15068 is based upon the knowledge of, and
63 documentation for, existing programs that utilize C-language interfaces. Any
64 questions regarding the definition of terms or the semantics of an underlying

65

66 1) The numbers in curly brackets correspond to those of the references in 1.2. When preceded by a
67 “B,” the numbers correspond to those of the bibliography in Annex A.

68 concept in this language should be referred to C Standard {B13}.

69 1.2 Normative References

70 The following standards contain provisions that, through references in this text,
71 constitute provisions of this part of ISO/IEC 15068. At the time of publication, the
72 editions indicated were valid. All standards are subject to revision, and parties to
73 agreements based on this part of ISO/IEC 15068 are encouraged to investigate the
74 possibility of applying the most recent editions of the standards listed below.
75 Members of IEC and ISO maintain registers of currently valid International Stan-
76 dards.

- 77 {1} ISO/IEC 646: 1991, *Information technology—ISO 7-bit coded character set*
78 *for information interchange* (International Reference Version).²⁾
- 79 {2} ISO/IEC 9945-1: 1996 (ANSI/IEEE Std 1003.1-1996), *Information*
80 *technology — Portable Operating System Interface (POSIX™) — Part 1: Sys-*
81 *tem Application Program Interface (API) [C Language]*.³⁾
- 82 {3} ISO/IEC 9945-2: 1993 (ANSI/IEEE Std 1003.2-1992), *Information*
83 *technology — Portable Operating System Interface (POSIX™) — Part 2:*
84 *Shell and Utilities*.
- 85 {4} ISO/IEC 10646-1: 1993, *Information technology—Universal Multiple-Octet*
86 *Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual*
87 *Plane*.

88 1.3 Conformance

89 1.3.1 Implementation Conformance

90 1.3.1.1 Conforming POSIX.7.2 Implementation

91 A Conforming POSIX.7.2 Implementation, also known as a “conforming imple-
92 mentation,” shall meet all the following criteria:

- 93 (1) The system shall support all interfaces defined within this part of ISO/IEC
94 15068. These interfaces shall support all the functional behavior
95 described herein. The interfaces covered by this definition of confor-
96 mance include, but are not limited to, utilities and their options and
97 extended options, the behavior of the utilities, including the generation of
98 events; events; structures; attributes and their values; and file formats.

99

100 2) ISO/IEC documents can be obtained from the ISO Central Secretariat, Case Postale 56, 1 rue de
101 Varembe, CH-1211, Genève 20, Switzerland/Suisse.

102 3) IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445
103 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

(2) The system may provide additional or enhanced utilities, functions, or facilities not required by this part of ISO/IEC 15068. Nonstandard extensions should be identified as such in the system documentation. Nonstandard extensions should conform to 2.10.2, of POSIX.2 {3}. Nonstandard extensions, when used, may change the behavior of utilities, functions, or facilities defined by this part of ISO/IEC 15068. In such cases, the conformance document for the implementation (see 2.2.1.3) shall define an execution environment (i.e., shall provide general operating instructions) in which a Strictly Conforming POSIX.7.2 Distribution may be operated upon and yield the behavior specified by this part of ISO/IEC 15068. In no case shall such an environment require modification of a Strictly Conforming POSIX.7.2 Distribution.

An implementation shall be a Conforming POSIX.7.2 Implementation if all the criteria for such are met with the exception of those features that depend on the existence of conforming implementations of either POSIX.1 {2} or POSIX.2 {3}. In this case, the conformance document for the Conforming POSIX.7.2 Implementation shall describe the behavior of the implementation of all features of the implementation, or of a Strictly Conforming POSIX.7.2 Distribution, that depend on the function of POSIX.1 {2} or POSIX.2 {3}. See 2.3.1 and 2.3.2.

1.3.1.2 Limited Conformance POSIX.7.2 Implementation

A Limited Conformance POSIX.7.2 Implementation shall meet all of the criteria established for a Conforming POSIX.7.2 Implementation (see 1.3.1.1) with the following exception:

— For the value of **HOST** in specifications of sources and targets (see 4.1.4.2), the system may support only the local machine. While this type of limited conformance removes support for remote operations, the syntax of all utilities and files shall remain identical to that required for Conforming POSIX.7.2 Implementations. The way in which this limitation is imposed by the implementation shall be implementation defined.

1.3.1.3 Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to this part of ISO/IEC 15068. The conformance document shall have the same structure as this part of ISO/IEC 15068, with the information presented in the appropriately numbered sections. Sections that consist solely of subordinate section titles, with no other information, are not required.

The conformance document shall not contain information about extended facilities or capabilities outside the scope of this part of ISO/IEC 15068, unless those extensions affect the behavior of a Strictly Conforming POSIX.7.2 Distribution; in such cases, the documentation required by the previous subclause shall be included.

The conformance document shall contain a statement that indicates the full name, number, and date of this part of ISO/IEC 15068 that applies. The conformance document may also list software standards approved by ISO/IEC or any ISO/IEC member body that are available for use by a Conforming POSIX.7.2 Implementation or by a Conforming POSIX.7.2 Distribution. Applicable characteristics where

documentation is required by one of these standards, or by standards of government bodies, may also be included.

The conformance document shall describe the behavior of the implementation for all implementation-defined features defined in this part of ISO/IEC 15068. This requirement shall be met by listing these features and providing either a specific reference to the system documentation or full syntax and semantics of these features. When the value or behavior in the implementation is designed to be variable or customizable on each instantiation of the system, the implementation provider shall document the nature and permissible ranges of this variation. When information required by this part of ISO/IEC 15068 is related to the underlying operating system and is already available in the POSIX.1 {2} or POSIX.2 {3} conformance document, the implementation need not duplicate this information in the conformance document for this part of ISO/IEC 15068, but may provide a cross-reference for this purpose.

The conformance document shall indicate whether the implementation is based on an underlying operating system that is fully conforming to both POSIX.1 {2} and POSIX.2 {3}.

If the implementation is not based on a conforming implementation of POSIX.1 {2}, then the conformance document shall describe the behavior of the implementation for all features of a Strictly Conforming POSIX.7.2 Distribution that depend on the function of POSIX.1 {2}. See 2.3.1.

If the implementation is not based on a fully conforming implementation of POSIX.2 {3}, then the conformance document shall describe the behavior of the implementation for all features of a Strictly Conforming POSIX.7.2 Distribution that depend on the function of POSIX.2 {3}. This dependency includes all of the utilities of POSIX.2 {3}, including the shell. See 2.3.2.

The conformance document may specify the behavior of the implementation for those features where this part of ISO/IEC 15068 states that implementations may vary or where features are identified as undefined or unspecified.

No specifications other than those described in this subclause (1.3.1.3) shall be present in the conformance document.

The phrase “shall be documented” in this part of ISO/IEC 15068 means that documentation of the feature shall appear in the conformance document, as described previously, unless the system documentation is explicitly mentioned.

The system documentation should also contain the information found in the conformance document.

1.3.1.4 Conforming Implementation Options

Additional utility options, distribution formats, or software structure attributes and values, may be provided in other related standards or in future revisions to this part of ISO/IEC 15068, without requiring this part of ISO/IEC 15068 to be updated.

188 **1.3.2 Distribution Conformance**

189 All distributions claiming conformance to this part of ISO/IEC 15068 fall within
190 one of the categories in the following subclauses.

191 **1.3.2.1 Strictly Conforming POSIX.7.2 Distribution**

192 A Strictly Conforming POSIX.7.2 Distribution is a distribution that requires only
193 the facilities described in this part of ISO/IEC 15068 (including any required facili-
194 ties of the underlying operating system; see 2.3). Such a distribution

- 195 (1) Shall contain only those files and directories defined in the Software
196 Packaging Layout (see Section 5)
- 197 (2) Shall contain only those software structures and attributes within those
198 structures described in this part of ISO/IEC 15068
- 199 (3) Shall use only the values of software structure attributes defined as valid
200 within described in this part of ISO/IEC 15068
- 201 (4) Shall use only the POSIX.2 {3} shell, **sh**, to invoke control files
- 202 (5) Shall not use facilities, structures, attributes, or values designated as
203 *obsolescent*

204 Within this part of ISO/IEC 15068, any restrictions placed upon a Conforming
205 POSIX.7.2 Distribution also shall restrict a Strictly Conforming POSIX.7.2 Distri-
206 bution.

207 **1.3.2.2 Conforming POSIX.7.2 Distribution**

208 A Conforming POSIX.7.2 Distribution is a distribution that uses only the facilities
209 described in this part of ISO/IEC 15068 and implied facilities of the underlying
210 operating system. See 2.3. Such a distribution shall use only the POSIX.2 {3}
211 shell, **sh**, to invoke control files. In addition to the software structures, and attri-
212 butes within those structures, allowable in a Strictly Conforming POSIX.7.2 Dis-
213 tribution, a Conforming POSIX.7.2 Distribution can also contain:

- 214 (1) Additional attributes in the manner supported by this part of ISO/IEC
215 15068.
- 216 (2) Additional files within the distribution, except where this part of ISO/IEC
217 15068 prohibits such files.

218 The term Conforming POSIX.7.2 Distribution is used to describe either of the two
219 following distribution types.

220 **1.3.2.2.1 ISO/IEC Conforming POSIX.7.2 Distribution**

221 An ISO/IEC Conforming POSIX.7.2 Distribution shall also include a statement of
222 conformance that documents all other ISO/IEC standards used.

223 **1.3.2.2.2 <National Body> Conforming POSIX.7.2 Distribution**

224 A <National Body> Conforming POSIX.7.2 Distribution differs from an ISO/IEC
Conforming POSIX.7.2 Distribution in that it also may use specific standards of a
single ISO/IEC member body referred to here as “<National Body>.” Such a

225 distribution shall include a statement of conformance that documents all other
226 <National Body> standards used.

227 **1.3.2.3 Conforming POSIX.7.2 Distribution Using Extensions**

228 A Conforming POSIX.7.2 Distribution Using Extensions is a distribution that
229 differs from a Conforming POSIX.7.2 Distribution only in that it either requires
230 behavior of an implementation other than that which is specified by this part of
231 ISO/IEC 15068, or requires invocation of control files with other than the
232 POSIX.2 {3} shell, or both.

233 NOTE: One example of non-standard behavior is that of requiring implementations to understand,
234 and take action based upon, the values of various vendor-defined attributes. Another example of
235 non-standard behavior is the ability to properly handle files with names drawn from other than the
236 portable filename character set. Vendors creating such distributions are encouraged to transform
237 filenames using UTF-8 {4}.

238 In addition to the documentation required of a Conforming POSIX.7.2 Distribu-
239 tion, such a distribution

240 (1) Shall fully document its requirements for these extended facilities,
241 whether required within the software packaging layout, within the utili-
242 ties defined in this part of ISO/IEC 15068, or both

243 (2) Shall document whether it is possible to convert this distribution into a
244 Conforming POSIX.7.2 Distribution, and what those steps are

245 (3) Shall document the extent to which the software contained in the distri-
246 bution can be managed by the utilities in accordance with this part of
247 ISO/IEC 15068

248 (4) Shall document the requirement for any interpreter other than the
249 POSIX.2 {3} Shell

250 A Conforming POSIX.7.2 Distribution Using Extensions shall be either an
251 ISO/IEC Conforming POSIX.7.2 Distribution Using Extensions, or a <National
252 Body> Conforming POSIX.7.2 Distribution Using Extensions. See 1.3.2.2.1 and
253 1.3.2.2.2.

254 **1.3.2.4 Documentation**

255 A conformance document with information required by 1.3.2 shall be available for
256 a distribution claiming conformance to this part of ISO/IEC 15068.

257 The conformance document shall contain a statement that indicates the full name,
258 number, and date of this part of ISO/IEC 15068 that applies. The conformance
259 document may also list software standards approved by ISO/IEC or any ISO/IEC
260 member body that are available for use by a Conforming POSIX.7.2 Implementa-
261 tion or by a Conforming POSIX.7.2 Distribution. Applicable characteristics where
262 documentation is required by one of these standards, or by standards of govern-
263 ment bodies, may also be included.

264 **1.4 Test Methods**

265 There are no specific test methods for this part of ISO/IEC 15068.

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 Editorial Conventions

This part of ISO/IEC 15068 uses the following editorial and typographical conventions. A summary of typographical conventions is shown in Table 2-1.

Table 2-1 – Typographical Conventions

Reference	Example
Attribute of a Structure (object)	<i>filesets</i> or <i>product.filesets</i>
C-Language Data Type	<i>long</i>
C-Language Function	<i>system()</i>
Cross Reference: Annex	Annex A
Cross Reference: Clause	2.3
Cross Reference: Other Standard	ISO/IEC 9999-1 { <i>n</i> }
Cross Reference: Section	Section 2
Cross Reference: Subclause	2.3.4, 2.3.4.5, 2.3.4.5.6
Defined Term	(see text)
Definition Citation	[POSIX.1 {2}]
Environment Variable	PATH
Error Number	[EINTR]
Event Status: Event Code	SW_ERROR: SW_IO_ERROR
Example Input	echo Hello, World
Example Output	Hello, World
Figure Reference	Figure 7-1
File Name	/tmp
Keyword	file_permissions
Parameter	<i><directory pathname></i>
Script Types	preinstall
Special Character	<newline>
Symbolic Constant, Limit	{_POSIX_VDISABLE}, {LINE_MAX}
Table Reference	Table 6-1
Utility Extended Option	<i>reinstall</i>
Utility Extended Option with Value	<i>reinstall=true</i>
Utility Name	awk
Utility Operand	<i>file_name</i>
Utility Option	-c
Utility Option with Option-Argument	-w width
Value of an Attribute	true

39 The **Bold** font is used to show brackets that denote optional arguments in a utility
40 synopsis, as in

41 `cut [-c list][file_name]`

42 These brackets shall not be used by the application unless they are specifically
43 mentioned as literal input characters by the utility description.

44 There are symbols enclosed in angle brackets (< >):

45 Parameters Parameters, also called *metavariables*, are in italics,
46 such as <*directory pathname*>. The entire symbol,
47 including the brackets, is meant to be replaced by the
48 value of the symbol described within the brackets.

49 In some examples, the **Bold Courier** font is used to indicate the output of the
50 system that resulted from some user input, shown in **Courier**.

51 Defined terms are shown in three styles, depending on context:

52 (1) Terms defined in 2.2.1, and 2.2.2 are expressed as subclause titles. Alter-
53 native forms of the terms appear in [brackets]. At the conclusion of this
54 type of definition, the designation of a standard within [brackets] indi-
55 cates that the text of the definition was copied from that standard, with
56 only editorial changes, if any.

57 (2) The initial appearances of other terms, applying to a limited portion of
58 the text, are in *italics*.

59 (3) Subsequent appearances of the term are in the Roman font.

60 Symbolic constants are shown in the following two styles — those within curly
61 brackets are intended to call the attention of the reader to values in <**limits.h**>
62 and <**unistd.h**>; those without braces are usually defined by one or a few
63 related functions. There is no semantic difference between these two forms of
64 presentation.

65 Events are shown within text in uppercase form and enclosed in parentheses.
66 This is done to call the attention of the reader to the values described in 4.1.6.2.

67 Filenames and pathnames are shown in **Courier**. When a pathname is shown
68 starting with **\$HOME/**, this indicates the remaining components of the pathname
69 are to be related to the directory named by the user's **HOME** environment
70 variable.

71 The style selected for some of the special characters, such as <**newline**>,
72 matches the form of the input given to the **localedef** utility (see 2.5.2 of
73 POSIX.2 {3}). Generally, the characters selected for this special treatment are
74 those that are not visually distinct, such as the control characters <**tab**> or
75 <**newline**>.

76 Literal characters and strings used as input or output are shown in various ways,
77 depending on context as follows:

78 %, **begin** When no confusion would result, the character or string is ren-
79 dered in the Courier font and used directly in the text.

80 **'c'** In some cases a character is enclosed in single-quote characters,
81 similar to a character constant used by several programming
82 languages. Unless otherwise noted, the quotes shall not be used
83 as input or output.

84 **"string"** In some cases, a string is enclosed in double-quote characters
85 similar to a string constant used by several programming
86 languages. Unless otherwise noted, the quotes shall not be used
87 as input or output.

88 Defined names that are usually in lowercase, particularly function names, are
89 never used at the beginning of a sentence or anywhere else that English usage
90 would require them to be capitalized.

91 Parenthetical expressions within normative text also contain normative informa-
92 tion. The general typographic hierarchy of parenthetical expressions is:

93 { [()] }

94 The square brackets are most frequently used to enclose a parenthetical expres-
95 sion that contains a function name [such as *waitpid()*] with its built-in
96 parentheses.

97 In some cases, tabular information is presented inline; in others it is presented in
98 a separately labeled table. This arrangement was employed purely for ease of
99 reference and there is no normative difference between these two cases.

100 Annexes marked as *normative* are parts of the standard that pose requirements,
101 exactly the same as the numbered sections, but have been moved to near the end
102 of the document for clarity of exposition. *Informative* annexes are for information
103 only and pose no requirements. All material preceding page 1 of the document
104 (the “front matter”) and the two indexes at the end are also only informative.

105 NOTES that appear in a smaller point size have one of two different meanings as
106 follows, depending on their location:

107 — When they are within the normal text of the document, they are the same
108 as footnotes — informative, posing no requirements on implementations,
109 distributions, or applications.

110 — When they are attached to tables or figures, they are normative, and can
111 include requirements.

112 Text marked as examples (including the use of “e.g.”) is for information only.

113 The table in this section refers to constructs from the C language. This part of
114 ISO/IEC 15068 itself has no requirement for the use of the C language or any other
115 programming language in particular. However, this part of ISO/IEC 15068 has
116 been developed with the use of concepts from POSIX.1 {2} and POSIX.2 {3}, and
117 those standards do make use of C-language constructs. The use of C-language
118 constructs within this part of ISO/IEC 15068 is simply to provide reference to the
119 appropriate parts of POSIX.1 {2} and POSIX.2 {3}. As far as this part of ISO/IEC
120 15068 is concerned, use of equivalent function from other languages is acceptable.

121 In some cases certain characters are interpreted as special characters by the
122 shell. In the normative portions of this part of ISO/IEC 15068, these characters
 are shown without escape characters or quoting (see 3.2 in POSIX.2 {3}). In all
 examples, however, quoting has been used, showing how sample commands

2.1 Conventions

(utility names combined with arguments) could be passed correctly to a shell (see **sh** in 4.56 of POSIX.2 {3}), or as a string to the *system()* function.

The typographical conventions listed here are for ease of reading only. Editorial inconsistencies in the use of typography are unintentional and have no normative meaning in this part of ISO/IEC 15068.

2.1.2 Grammar Conventions

Portions of this part of ISO/IEC 15068 are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the **yacc** utility (see A.3 in POSIX.2 {3}). However, it does not represent fully functional **yacc** input suitable for program use. The lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial **yacc** code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

The following typographical conventions are used in the grammar; they have no significance except to aid in reading.

- The identifiers for the reserved words of the language are shown with a leading capital letter. These are terminals in the grammar. Examples: **while**, **case**.
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores. Examples: **NEWLINE**, **ASSIGN_OP**, **NAME**.
- The identifiers for nonterminals are all lowercase.

2.2 Definitions

2.2.1 Terminology

For the purposes of this part of ISO/IEC 15068, the following definitions apply:

2.2.1.1 application: Includes both humans and executable programs that use implementations of this part of ISO/IEC 15068. Such executable programs can include control files within a distribution.

2.2.1.2 can: An indication of a permissible optional feature or behavior available to a distribution or to someone using a utility defined in this part of ISO/IEC 15068; the implementation shall support such features or behaviors as mandatory requirements.

159 **2.2.1.3 conformance document:** A document provided by an implementor that
160 contains implementation details as described in 1.3.1.3.

161 NOTE: See 2.2.1.3 of POSIX.2 {3}.

162 **2.2.1.4 implementation:** An object providing, to distributions and users, the
163 services defined by this part of ISO/IEC 15068.

164 The word *implementation* is to be interpreted to mean that object, after it has
165 been modified in accordance with the manufacturer's instructions to:

- 166 — Configure it for conformance with this part of ISO/IEC 15068;
- 167 — Select some of the various optional facilities described by this part of
168 ISO/IEC 15068, through customization by local system administrators or
169 operators.

170 An exception to this meaning occurs when discussing conformance documentation
171 or using the term *implementation defined*.

172 NOTE: See 2.2.1.5 and 1.3.1.3.

173 **2.2.1.5 implementation defined:** An indication that the implementation pro-
174 vider shall define and document the requirements for correct program constructs
175 and correct data of a value or behavior.

176 When the value or behavior in the implementation is designed to be variable or
177 customizable on each instantiation of the system, the implementation provider
178 shall document the nature and permissible ranges of this variation.

179 NOTE: See 1.3.1.3. See also 2.2.1.5 of POSIX.2 {3}.

180 **2.2.1.6 may:** An indication of an optional feature or behavior of the implementa-
181 tion that is not required by this part of ISO/IEC 15068, although there is no prohi-
182 bition against providing it.

183 A Strictly Conforming POSIX.7.2 Distribution is permitted to use such features,
184 but shall not rely on the implementation's actions in such cases. To avoid ambi-
185 guity, the reverse sense of *may* is not expressed as *may not*, but as *need not*.

186 **2.2.1.7 obsolescent:** An indication that a certain feature may be considered for
187 withdrawal in future revisions of this part of ISO/IEC 15068.

188 NOTE: See 2.2.1.7 of POSIX.2 {3}.

189 **2.2.1.8 shall:** An indication of a requirement on the implementation or on
190 Strictly Conforming POSIX.7.2 Distributions, where appropriate.

191 **2.2.1.9 should:**

- 192 (1) With respect to implementations, an indication of an implementation
193 recommendation, but not a requirement.
- (2) With respect to distributions, an indication of a recommended practice for
distributions, and a requirement for Strictly Conforming POSIX.7.2

2.2 Definitions

194 Distributions.

195 **2.2.1.10 system documentation:** All documentation provided with an imple-
196 mentation, except the conformance document.

197 Electronically distributed documents for an implementation are considered part of
198 the system documentation.

199 NOTE: See 2.2.1.10 of POSIX.2 {3}.

200 **2.2.1.11 undefined:** An indication that this part of ISO/IEC 15068 imposes no
201 portability requirements on applications for erroneous program construction,
202 erroneous data, or use of an indeterminate value.

203 Implementations (or other standards) may specify the result of using that value or
204 causing that behavior. An application using such behaviors is using extensions, as
205 defined in 1.3.2.3.

206 NOTE: See 2.2.1.11 of POSIX.2 {3}.

207 **2.2.1.12 unspecified:** An indication that this part of ISO/IEC 15068 imposes no
208 portability requirements on applications for a correct program construction or
209 correct data.

210 Implementations (or other standards) may specify the result of using that value or
211 causing that behavior. An application requiring a specific behavior, rather than
212 tolerating any behavior when using that functionality, is using extensions, as
213 defined in 1.3.2.3.

214 NOTE: See 2.2.1.12 of POSIX.2 {3}.

215 **2.2.2 General Terms**

216 For the purposes of this part of ISO/IEC 15068, the following definitions apply:

217 **2.2.2.1 absolute path:** If the underlying system is based upon a conforming
218 implementation of POSIX.1 {2}; then a pathname that begins with /; otherwise,
219 *absolute path* is implementation defined.

220 **2.2.2.2 alternate root directory:** A pathname other than / for managing
221 installed software.

222 **2.2.2.3 analysis phase:** The steps a software administration utility performs,
223 before modifying the target, while attempting to ensure that the execution of
224 operations on the target will succeed.

225 **2.2.2.4 attribute:** A component of an object, possessing a name and one or more
226 values.

227 **2.2.2.5 autorecovery:** The process of restoring installed software to the state it
228 was in prior to the invocation, and subsequent failure during execution, of the
229 **swinstall** utility.

230 **2.2.2.6 autoselect:** The automatic selection, within a utility, of software beyond
231 that directly specified by the user in order to meet the dependencies of the user-
232 specified software.

233 **2.2.2.7 bundle:** A software object, which is a grouping of other software objects,
234 such as all or parts of other bundles and products.

235 NOTE: See 3.8.

236 **2.2.2.8 catalog:** The metadata describing all the software objects that are a part
237 of a single software_collection (distribution or installed_software object).

238 Catalogs exist both in distributions and for installed software, although storage of
239 catalogs for installed software is undefined within this part of ISO/IEC 15068.

240 A catalog in a distribution shall always use the exported catalog structure, since it
241 is required to be stored in a portable or exported catalog structure. A catalog for
242 installed software shall use the exported catalog structure when information is
243 listed with **swlist -v**.

244 NOTE: See Section 5.

245 **2.2.2.9 class:** Describes the structure and attributes of each level of the software
246 hierarchy that is used to organize and manage software files.

247 **2.2.2.10 client role:** The location where the software is actually executed or
248 used (as opposed to the target where it is actually installed).

249 The configuration of software is performed by this role.

250 **2.2.2.11 command line interface:** A means of invoking utilities by issuing com-
251 mands from within a POSIX.2 {3} shell, implying that neither graphics nor win-
252 dows are required.

253 **2.2.2.12 common class:** Defines those aspects of different software objects that
254 are the same.

255 The common classes for this part of ISO/IEC 15068 are software_collections,
256 software, and software_files. The names of these classes are also used to generi-
257 cally describe any object that shares that common class.

258 **2.2.2.13 compressed file:** A file that has been transformed in a manner
259 intended to reduce its size without loss of information.

260 **2.2.2.14 containment:** A relationship between two objects such that one is said
to belong to, or form part of, the other.

261 All objects except `software_collection` objects shall be contained within exactly one
262 object. The containment of `software_collection` objects is undefined within this
263 part of ISO/IEC 15068.

264 **2.2.2.15 control directory:** The directory below which the `control_files` for
265 filesets and products are stored within exported catalogs for distributions and
266 installed software.

267 **2.2.2.16 control_files:** The control scripts executed by the utilities, the INFO
268 file describing the files in a fileset, and other files associated with a software
269 object.

270 **2.2.2.17 control script:** A `control_file` associated with a software object that is
271 executed by the software administration utilities.

272 **2.2.2.18 corequisite:** The specification in a software object such that another
273 software object shall be installed, in conjunction with the installation of the first,
274 and configured in conjunction with the configuration of the first.

275 **2.2.2.19 decimal character string:** A sequence of characters from the set of
276 decimal digits the first of which shall not be the digit zero.

277 Decimal character strings shall consist only of the following characters:

278 0 1 2 3 4 5 6 7 8 9

279 Within software definition files of exported catalogs, all such strings shall be
280 encoded using IRV {1}.

281 **2.2.2.20 default option:** The value for an extended option as defined in a
282 defaults file.

283 NOTE: See 2.2.2.21 and 2.2.2.35.

284 **2.2.2.21 defaults file:** A system-specific or user-specific file that contains the
285 default values for extended options used by the software administration utilities.

286 **2.2.2.22 dependency:** A software object that is a prerequisite, corequisite or
287 exrequisite for a software object defining a **dependency_spec**.

288 A dependency is the object upon which another object depends.

289 **2.2.2.23 dependency_spec:** A **software_spec** that describes a dependency.

290 NOTE: See 4.1.4.1.

291 **2.2.2.24 dependent:** A software object that specifies a prerequisite, corequisite
292 or exrequisite on another software object.

2.2.2.25 developer role: Where software is developed, tested, and maintained.

293 This role is outside the scope of this part of ISO/IEC 15068.

294 **2.2.2.26 directory medium:** A medium that contains a distribution in a
295 POSIX.1 {2} hierarchical file system format.

296 NOTE: An example of this is a distribution contained in a POSIX.1 {2} file system format on a CD-
297 ROM.

298 **2.2.2.27 distribution:** A software_collection containing software in the software
299 packaging layout.

300 **2.2.2.28 distribution catalog:** The catalog of metadata for a distribution
301 software_collection.

302 Unlike a catalog for an installed_software object, a distribution catalog is stored in
303 a particular exported catalog structure that is part of the software packaging lay-
304 out.

305 **2.2.2.29 distribution path:** The pathname below which the catalog describing
306 the distribution is located.

307 If the distribution is on a single medium, all software for it is located below this
308 path.

309 **2.2.2.30 downdate:** Installation of software with a revision older than that of
310 the software currently installed in the same location.

311 This is also referred to as downgrading or reverting.

312 **2.2.2.31 event:** An occurrence that may require reporting by the utilities defined
313 in this part of ISO/IEC 15068.

314 The reporting of an event may cause data to be written to stdout, stderr, or to a
315 log file.

316 **2.2.2.32 execution phase:** The operations a software administration utility per-
317 forms that modify the target.

318 **2.2.2.33 exported catalog:** Refers to information organized in the exported
319 catalog structure of the standard packaging layout.

320 It is used for distribution catalogs as well as exporting installed software catalogs
321 using **swlist -c catalog**.

322 Within software definition files of an exported catalog, all data that can be
323 encoded using IRV {1}, shall be. Any such data that cannot be so encoded shall be
324 transformed using UTF-8 {4}.

325 NOTE: See Section 5.

326 **2.2.2.34 exrequisite:** The specification in a software object such that it shall not
327 be installed if one or more specific software objects are installed.

328 **2.2.2.35 extended option:** The options that can be specified with the **-x** option.
329 These options may be defined in defaults files or options files.

330 **2.2.2.36 filename:** A POSIX.1 {2} filename with characters drawn from the
331 POSIX.1 {2} portable filename character set.

332 NOTE: See 2.2.2.60 of POSIX.1 {2}.

333 **2.2.2.37 filename character string:** A sequence of characters from the portable
334 filename character set, not including the / (slash) character.

335 Within software definition files of exported catalogs, all such strings shall be
336 encoded using IRV {1}.

337 NOTE: See 2.2.2.60 of POSIX.1 {2}.

338 **2.2.2.38 fileset:** Defines the files that make up a software object, and is the
339 lowest level of software object that can be specified as input to the software
340 administration utilities.

341 NOTE: See 3.9.

342 **2.2.2.39 file storage structure:** The storage directories in the software packag-
343 ing layout under which the actual software files for each fileset are located.

344 NOTE: See Section 5.

345 **2.2.2.40 fully qualified software_spec:** A **software_spec** that always
346 identifies a software object unambiguously.

347 NOTE: See 4.1.4.1.

348 **2.2.2.41 graphical user interface:** A means of presenting function to a user
349 through the use of graphics.

350 All such interfaces are outside the scope of this part of ISO/IEC 15068.

351 **2.2.2.42 hard link:** A directory entry, as defined in 2.2.2.17. of POSIX.1 {2}.

352 **2.2.2.43 hexadecimal character string:** A sequence of characters from the set
353 of hexadecimal digits, preceded by the two characters **0x** (zero followed by a lower-
354 case "x").

355 Hexadecimal character strings shall consist only of the following characters:

356 **0 1 2 3 4 5 6 7 8 9 A B C D E F x**

357 Within software definition files of exported catalogs, all such strings shall be
encoded using IRV {1}.

358 **2.2.2.44 host:** A machine that contains software managed by this part of ISO/IEC
359 15068.

360 NOTE: A host may contain both `installed_software` and `distribution software_collections`. The
361 name of the host is the starting point for finding all software on that machine managed by this part
362 of ISO/IEC 15068. The *path* attribute of a `software_collection`, along with the specification of a host,
363 can be used on the command line to identify a particular `software_collection` to be managed by this
364 part of ISO/IEC 15068.

365 **2.2.2.45 host character string:** A sequence of characters describing a host, as
366 defined in 2.2.2.44.

367 Within software definition files of an exported catalog, all data that can be
368 encoded using IRV {1} shall be. Any such data that cannot be so encoded shall be
369 transformed using UTF-8 {4}.

370 **2.2.2.46 INDEX file:** The file within an exported catalog containing the meta-
371 data describing the software objects and attributes for all bundles, products, sub-
372 products and filesets.

373 NOTE: The format of this file is defined in 5.2.

374 **2.2.2.47 INFO file:** For each product and fileset, the file within an exported cata-
375 log containing the metadata describing the `software_file` objects and attributes.

376 NOTE: The format of this file is defined in 5.2.

377 **2.2.2.48 inheritance:** The way in which the attribute definitions of a common
378 object class are used as a part of the definition of other object classes.

379 The definition of the new object class includes the definition of the common class
380 plus the additional definitions specific to the new object class.

381 **2.2.2.49 installed software:** Any software object created by the use of the
382 `swinstall` utility.

383 **2.2.2.50 installed_software:** A `software_collection` containing installed
384 software.

385 This software is in a state ready for use, or ready to be shared by client systems.
386 A directory path on a system and an `installed_software` catalog together identify a
387 unique `installed_software` object.

388 **2.2.2.51 installed_software catalog:** The catalog of metadata for an
389 `installed_software software_collection`.

390 Unlike a catalog for a distribution object, the storage and format of an
391 `installed_software` catalog is undefined within this part of ISO/IEC 15068. The
392 ability to dump and restore all or part of an `installed_software` catalog into an
exported catalog structure is included in this part of ISO/IEC 15068.

393 **2.2.2.52 installed_software path:** The root directory of an installed_software
394 object; the pathname below which all software for that object shall be installed.

395 **2.2.2.53 integer character string:** A decimal character string, an octal charac-
396 ter string, or a hexadecimal character string.

397 NOTE: See 2.2.2.19, 2.2.2.63, and 2.2.2.43.

398 **2.2.2.54 interactive:** The behavior of a utility or control_script which requires
399 input from the user during its execution.

400 **2.2.2.55 kernel:** The nucleus of the operating system.

401 NOTE: See B.2.2.2 of POSIX.1 {2}.

402 **2.2.2.56 kernel fileset:** A fileset in which one or more of the referenced files
403 forms part of the kernel, and denoted by having the value of its *is_kernel* attribute
404 set to **true**.

405 **2.2.2.57 locatable fileset:** A fileset for which permission is granted to install the
406 files in a different location as specified by the user, and denoted by having the
407 value of its *is_locatable* attribute set to **true**.

408 **2.2.2.58 locatable software:** Software that contains locatable filesets.

409 **2.2.2.59 manager role:** Where each task is initiated.

410 The *manager role* is concerned with taking appropriate action at the completion or
411 failure of a task.

412 **2.2.2.60 metadata:** The information kept about software.

413 It consists of the values of the various attributes of each of the objects.

414 **2.2.2.61 newline string:** A white space string consisting only of the **<newline>**
415 character.

416 NOTE: The term *white space string* is defined in 2.2.2.110 and **<newline>** character is defined in
417 2.2.2.107 of POSIX.2 {3}.

418 **2.2.2.62 object:** An instance in the software hierarchy that can be operated on
419 using the software administration utilities.

420 **2.2.2.63 octal character string:** A sequence of characters from the set of octal
421 digits the first of which shall be the digit zero.

422 Octal character strings shall consist only of the following characters:

423 **0 1 2 3 4 5 6 7**

Within software definition files of exported catalogs, all such strings shall be
encoded using IRV {1}.

424 **2.2.2.64 options file:** A file that can be specified with the `-x` option. This file
425 contains extended option definitions that override default definitions.

426 NOTE: See 2.2.2.21.

427 **2.2.2.65 packager role:** Where software that has been developed is organized in
428 a form suitable for distribution.

429 **2.2.2.66 pathname:** A POSIX.1 {2} pathname with characters drawn from the
430 POSIX.1 {2} portable character set.

431 NOTE: See 2.2.2.60 of POSIX.1 {2}.

432 **2.2.2.67 pathname character string:** A sequence of characters from the port-
433 able filename character, including the `/` (slash) character.

434 NOTE: See 2.2.2.60 of POSIX.1 {2}.

435 Within software definition files of exported catalogs, all such strings shall be
436 encoded using IRV {1}.

437 **2.2.2.68 portable character string:** A sequence of characters from the portable
438 character set as defined in 2.2.2.130 and 2.4 of POSIX.2 {3}.

439 Within software definition files of exported catalogs, all such strings shall be
440 encoded using IRV {1}.

441 **2.2.2.69 prerequisite:** The specification in a software object that implies it shall
442 not be installed until after some other software object is installed, and configured
443 until after the other software object is configured.

444 NOTE: The manner of honoring such a prerequisite is described in 4.5.7.1 and 4.5.7.2.

445 **2.2.2.70 product:** A software object used to define a set of related software.
446 Filesets are contained within products.

447 **2.2.2.71 product specification file (PSF):** The input file used to define the
448 structure and attributes of software objects and related files to be packaged by the
449 **swpackage** utility.

450 **2.2.2.72 proxy install:** A proxy install uses an alternate root directory as the
451 target path.

452 **2.2.2.73 recovery:** The ability of the **swinstall** utility, for a failed software
453 install, to return the system to the state that it was in before the failure, including
454 restoring the files.

455 **2.2.2.74 reboot fileset:** A fileset which, if installed, requires reboot of the
operating system to complete its installation, and denoted by having the value of
its *is_reboot* attribute set to **true**.

456 **2.2.2.75 rebooting:** An implementation-defined procedure generally used to ter-
457minate and then restart operations on the target system.

458 **2.2.2.76 role:** The context in which an operation is executed.

459 The utilities in this part of ISO/IEC 15068 require the ability to perform operations
460 on more than one system, perhaps by more than one person. These operations are
461 separated into distinct roles including developer, packager, manager, source, tar-
462get, and client.

463 **2.2.2.77 selection phase:** The set of steps performed by software administration
464utility to process selections and options.

465 **2.2.2.78 serial medium:** A medium that contains a POSIX.1 {2} extended **tar** or
466extended **cpio** archive.

467 NOTE: See 10.1.1 and 10.1.2 of POSIX.1 {2}.

468 **2.2.2.79 session:** An execution of a software administration command from ini-
469tiation to completion on all applicable roles.

470 **2.2.2.80 shell token string:** A sequence of shell tokens.

471 A shell token string shall be a portable character string.

472 NOTE: See 2.2.2.68. Shell tokens are defined in 3.3 of POSIX.2 {3}.

473 **2.2.2.81 software:** A generic term referring to software objects or a structured
474set of files.

475 This term can refer to the objects forming the hierarchical structure (software
476objects), or to the actual files and control_files (software files).

477 NOTE: See 2.2.2.90.

478 **2.2.2.82 software_collection:** A grouping of software objects that are managed
479by the software administration utilities.

480 Software_collections are the sources and targets of these utilities. This part of
481 ISO/IEC 15068 defines two types of software_collections: installed_software and
482distributions.

483 **2.2.2.83 software common class:** The common class describing the common
484attributes associated with the hierarchical structure of software objects defined by
485this part of ISO/IEC 15068.

486 **2.2.2.84 software definition files:** The files containing the software structure
487and detailed attributes for distributions, installed_software, bundles, products,
488subproducts, filesets, files, and control_files.

This includes the **INDEX** and **INFO** files and the PSF.

489 To communicate metadata information relating to both distributions and installed
490 software, software definition files serve as input to, or output from, the various
491 software administration utilities. The format used by software administration
492 utilities to store metadata relating to installed software is undefined.

493 NOTE: See 5.2.

494 **2.2.2.85 software file:** A generic term referring to the files and control_files that
495 are contained within software objects and managed by the utilities in this part of
496 ISO/IEC 15068.

497 **2.2.2.86 software_file common class:** The common class that relates the two
498 types of files defined by this part of ISO/IEC 15068, namely the actual files that
499 make up the software, plus the control_files that are executed by the utilities
500 when operating on software.

501 **2.2.2.87 software_files:** A generic term referring to file and control_file objects
502 (those that share the same software_file common class).

503 **2.2.2.88 software hierarchy:** Hierarchical organization of objects that are
504 managed by the software administration utilities.

505 **2.2.2.89 software location:** The directory relative to the installed_software root
506 directory where the relocatable files of the software have been located.

507 **2.2.2.90 software object:** An object that inherits attributes of the software com-
508 mon class, meaning a bundle, product, subproduct, or fileset object.

509 **2.2.2.91 software packaging layout:** The format for software in a distribution.
510 It contains the metadata for the distribution catalog in a well-defined exported
511 form, as well as the files for the software objects in that distribution.

512 NOTE: For a detailed description, see Section 5.

513 **2.2.2.92 software pattern match string:** A sequence of one or more strings,
514 each made up of a sequence of one or more characters from the shell “Pattern
515 Matching Notation” strings described in 3.13, of POSIX.2 {3}, and with the mean-
516 ing defined in that clause. If there are two or more strings, the strings are
517 separated by the | character.

518 The match is true if any of the sequences of strings match according to 3.13 of
519 POSIX.2 {3}.

520 A software pattern match string shall be portable character string.

521 **2.2.2.93 software_spec:** A string that is used to identify one or more software
522 objects for input to a software administration utility.

523 NOTE: See 4.1.4.1.1.

524 **2.2.2.94 source:** The specification of a source distribution object for a software
525 administration utility. The source host provides a means to locate the source role
526 and the source path is a path accessible to the source host.

527 **2.2.2.95 source host:** The host portion of a source specification.

528 **2.2.2.96 source path:** The pathname portion of a source specification.

529 **2.2.2.97 source role:** Where the software exists in a form suitable for distribu-
530 tion, forming a context for the establishment of a repository of software from
531 which the manager may choose to distribute to targets.

532 Software exists in the source until it is removed by a task initiated by the
533 manager. The source role provides a repository where software may be stored and
534 provides access for those roles that require the software.

535 **2.2.2.98 subproduct:** A software object that is a grouping of software filesets
536 and other subproducts within a product.

537 NOTE: See 3.10.

538 **2.2.2.99 symbolic link:** A type of file that contains a pathname.

539 Rather than containing data itself, this type of file will resolve to another, as
540 defined by the contained pathname. The way in which this type of file is handled
541 by implementations of this part of ISO/IEC 15068 is undefined.

542 NOTE: It is not the intention of this part of ISO/IEC 15068 to define symbolic links in a manner
543 inconsistent with POSIX.1 {2}. However, no approved POSIX standard currently contains symbolic
544 links. This definition is a placeholder until such time as an approved standard provides the
545 definition. See POSIX.1a {B21}.

546 **2.2.2.100 system:** An implementation of this part of ISO/IEC 15068.

547 **2.2.2.101 target:** The specification of a target distribution object, or installed
548 software object, for a software administration utility. The target host provides a
549 means to locate the target role and the target path is a path accessible to the tar-
550 get host.

551 **2.2.2.102 target host:** The host portion of a target specification.

552 **2.2.2.103 target path:** The pathname portion of a target specification.

553 **2.2.2.104 target role:** Where software is installed, removed, listed, and other-
554 wise operated on by the utilities.

555 For example, when installing software, the target is where software is installed
556 after having been delivered from a source. As another example, the target for a
copy operation command refers to the distribution to which products are added.
For management operations like removing software, the target refers to either the

557 installed_software objects or the distributions from which software is being
558 removed.

559 **2.2.2.105 update:** Installing a newer revision of software than one that is
560 currently installed, into the same location.

561 This is also referred to as upgrading.

562 **2.2.2.106 vendor:** A supplier of packaged software.

563 This term applies to anyone who creates packaged software, including commercial
564 and non-commercial suppliers, system administrators, and end users.

565 **2.2.2.107 vendor-defined:** An item, such as a nonstandard attribute, that is
566 defined by the vendor that created (packaged) the software.

567 **2.2.2.108 vendor-supplied:** An item, such as a control file, that is supplied by
568 the creator (packager) of the software.

569 **2.2.2.109 version:** A unique identification of software based on the attributes of
570 the software. Version differentiates software objects with the same value of the
571 *tag* attribute.

572 Versions of bundles or products have the same value of the *tag* attribute and will
573 differ by the value of at least one of *revision*, *architecture*, *vendor_tag*, *location*, or
574 *qualifier* attributes. The *location* and *qualifier* attributes only apply to software in
575 installed_software software_collections.

576 A fileset is considered a version of another fileset if they have the same *fileset.tag*
577 and their respective products have the same *product.tag*.

578 **2.2.2.110 white space string:** A sequence of one or more white space characters
579 (as defined in 2.2.2.191 of POSIX.2 {3}) including **<space>**, **<tab>**, and **<new-**
580 **line>**.

581 Within software definition files of exported catalogs, all such strings shall be
582 encoded using IRV {1}.

583 **2.2.2.111 wildcard character:** One of *?[(asterisk, question mark, open
584 bracket).

585 Such characters are used in software pattern match strings 2.2.2.92.

586 2.2.3 Abbreviations

587 For the purposes of this part of ISO/IEC 15068, the following abbreviations apply:

588 **2.2.3.1 API:** Application Programming Interface

- 589 **2.2.3.2 CLI:** Command Line Interface, as defined in 2.2.2.11.
- 590 **2.2.3.3 C Standard:** ISO/IEC 9899: 1990, *Information technology—*
591 *Programming languages—C* {B13}.
- 592 **2.2.3.4 CRC:** Cyclic Redundancy Check.
- 593 **2.2.3.5 GUI:** Graphical User Interface, as defined in 2.2.2.41.
- 594 **2.2.3.6 IRV:** ISO/IEC 646: 1991, *Information technology—ISO 7-bit coded char-*
595 *acter set for information interchange*, International Reference Version {1}.
- 596 **2.2.3.7 newline:** A newline string, as defined in 2.2.2.61.
- 597 **2.2.3.8 OS:** Operating System.
- 598 **2.2.3.9 POSIX.1:** ISO/IEC 9945-1: 1990 (ANSI/IEEE Std 1003.1-1990): *Informa-*
599 *tion technology—Portable Operating System Interface (POSIX®)—Part 1: System*
600 *Application Program Interface (API) [C Language]* {2}.
- 601 **2.2.3.10 POSIX.1a:** IEEE P1003.1a, *Draft Standard for Information*
602 *technology—Portable Operating System Interface (POSIX®)—Part 1: System Appli-*
603 *cation Program Interface (API) [C Language]* {B21}.
- 604 **2.2.3.11 POSIX.2:** ISO/IEC 9945-2: 1993 (ANSI/IEEE Std 1003.2-1992): *Infor-*
605 *mation technology—Portable Operating System Interface (POSIX®)—Part 2: Shell*
606 *and Utilities* {3}.
- 607 **2.2.3.12 POSIX.7.2:** This part of ISO/IEC 15068.
- 608 **2.2.3.13 PSF:** product specification file as defined in 2.2.2.71.
- 609 **2.2.3.14 symlink:** symbolic link, as defined in 2.2.2.99.
- 610 **2.2.3.15 UTF-8:** UCS Transformation Format 8, as defined in ISO/IEC 10646-
611 1: 1993, *Information technology—Universal Multiple-Octet Coded Character Set*
612 *(UCS)—Part 1: Architecture and Basic Multilingual Plane, Amendment 2: UCS*
613 *Transformation Format 8 (UTF-8), 1996* {4}.
- 614 **2.2.3.16 white space:** A white space string, as defined in 2.2.2.110.

2.3 Dependencies on Other Standards

2.3.1 Features Inherited From POSIX.1

This subclause describes some of the features provided by POSIX.1 {2} that are assumed to be globally available to all conforming implementations. This subclause does not attempt to detail all the POSIX.1 {2} features that are required by all the utilities defined in this part of ISO/IEC 15068; the utility descriptions point out additional functionality required to provide the corresponding features needed.

The following subclauses describe frequently used concepts. Utility description statements override these defaults when appropriate.

2.3.1.1 File System

The hierarchical directory structure of POSIX.1 {2} is assumed to be available, as well as support for case-sensitive file names. In addition, various file attributes are also assumed to be present, including the following — type, owner, group, mode, uid, gid, mtime, major, and minor.

2.3.1.2 Environment Variables

The existence of environment variables in general is assumed, as well as **PATH**, **LANG**, **LC_ALL**, **LC_CTYPE**, **LC_MESSAGES**, **LC_TIME**, and **TZ**, in particular.

2.3.1.3 Data Interchange Format

The ability to read and write the data interchange formats of POSIX.1 {2} is assumed, including both extended **tar** and extended **cpio**. See 10.1.1 and 10.1.2 of POSIX.1 {2}. See also 5.3.

2.3.2 Features Inherited From POSIX.2

This subclause describes some of the features provided by POSIX.2 {3} that are assumed to be globally available to all systems conforming to this part of ISO/IEC 15068. This subclause does not attempt to detail all of the POSIX.2 {3} features that are required by all the utilities and control scripts defined in this part of ISO/IEC 15068; additional functionality required may be found in the utility descriptions and in 4.1.6.1.

All of the utilities defined in POSIX.2 {3} are required, including the shell interpreter (**sh**). This assures a portable environment for executable control files.

Section 3: Software Structures

This section describes the software classes and attributes applicable to software administration. Each utility in Section 4 describes the operations on the software objects including how the values of the attributes affect the behavior of the operations. Whether these operations and behaviors are implemented as procedures on software structures or by other means is undefined within this part of ISO/IEC 15068.

The software administration classes form a hierarchy that consists of distributions, media, installed_software, vendors, bundles, products, subproducts, filesets, control_files, and files.

At each level, this hierarchy is defined by containment attributes that reference objects at lower levels. Operations on objects of lower levels, such as files, are actually enacted by operations on objects of higher levels. For example, files may be created in a distribution by copying a software product.

A “common class” is used to define attributes that are common between related objects. Objects inherit attribute definitions from common classes as well as their individual attributes. This provides a logical relationship between the objects that share the same common class. The software administration common classes are software_collection, software, and software_file.

Objects that share the same common class are also referred to generically as software_collections, software objects, and software_files.

In tables in this section, attributes are listed with various properties. The attributes and their values manifest themselves as part of the utilities defined in Section 4 and the software packaging layout in Section 5.

The names of attributes are as provided. If the underlying host allows for the distinction of case, the attribute names shall be sensitive to case. Where values of attributes are shown, if the underlying host allows for the distinction of case, the values of attributes shall be sensitive to case. If the underlying host does not allow for the distinction of case for either the name or value of an attribute, the way in which case differences are handled is implementation defined.

The attribute tables in this section list the following information:

Attribute

The name of the attribute, also used as the keyword for the attribute.

Length

The maximum permitted length of the value of the attribute.

All attribute values in this part of ISO/IEC 15068 are represented only as strings. The length is the maximum permitted length of the value in bytes or, for attributes whose values are lists, the maximum permitted number of items permitted in the list. Since the means of

37 storing such data for installed software is undefined within this part
38 of ISO/IEC 15068, an implementation may store such values inter-
39 nally in different structures for installed software. See 2.2.2.28,
40 2.2.2.33, and 2.2.2.51.

41 *Permitted Values*

42 The character sequences permitted as values for this attribute.

43 *Default Value*

44 The value of the attribute if the attribute is not specified.

45 A default value of **None** means the system shall not supply a value in
46 cases where the attribute has not been specified and the attribute is
47 not one whose values are generated dynamically. See 5.1.1 and 5.2.

48 The attribute tables are broken into the following three groups:

- 49 (1) The top group contains the attributes that are used to identify a particu-
50 lar instance.
- 51 (2) The middle group contains the rest of the attributes that describe other
52 information or behaviors for the object.
- 53 (3) The bottom group contains the attributes that describe the objects con-
54 tained within this object. The way in which these lists are represented in
55 software definition files is described in 5.2. The way in which these lists
56 are represented by **swlist** is described in 4.6.3.

57 Beyond this convention, the order of attributes shown in this section is not
58 significant. For any attribute ordering rules, see 5.2. Some attributes do not
59 apply to software objects in both distributions and installed_software objects. See
60 5.2 for details.

61 Management of lists of software_collections contained within a host is undefined
62 within this part of ISO/IEC 15068. See 4.1.4.2 for the way in which
63 software_collections are identified relative to a software host.

64 **3.1 Software_Collection**

65 A software_collection is the common class from which distribution and
66 installed_software objects inherit.

67 A software_collection can contain product and bundle software objects. A
68 software_collection can contain multiple versions of the same product or bundle
69 software objects, namely products or bundles that share the same value for the *tag*
70 attribute.

71 Each software_collection has a catalog associated with it that contains the meta-
72 data describing all software objects in that collection.⁴⁾

73 NOTE: For distribution software_collections, the catalog information is stored in the software
74 packaging layout in an exported catalog structure. For installed_software objects, how the catalog
75 information is stored (whether in a file or database, for example) is undefined within this part of
76 ISO/IEC 15068.

Table 3-1 – Attributes of the Software_Collection Common Class

Attribute	Length	Permitted Values	Default Value
<i>path</i>	Undefined	Pathname character string	Implementation defined
<i>dfiles</i>	64	Filename character string	dfiles
<i>layout_version</i>	64	1.0	1.0
<i>pfiles</i>	64	Filename character string	pfiles
<i>bundles</i>	Undefined	List of bundle_software_specs	Empty list
<i>products</i>	Undefined	List of product_software_specs	Empty list

3.1.1 Software_Collection Attributes

The attributes listed in Table 3-1 and described in the following, characterize each instance of the software_collection class, and are inherited by each instance of the distribution and installed_software classes:

- bundles* A list of **bundle_software_specs**.
Each **software_spec** shall refer to a bundle. Each **software_spec** shall be fully qualified. See 4.1.4.1 for the syntax of **software_spec**.
- dfiles* The name of the directory in the exported catalog structure below which any attributes stored as files for the software_collection are stored (see 5.1).
- layout_version* This attribute, and its value, are included for future use.
- path* The identifier for a particular software collection on a host.
The value of the path attribute shall be an absolute path. The default value of this attribute is implementation defined. See 4.1.5.2.
- pfiles* The name of the directory in the exported catalog structure below which any control_files, and attributes stored as files, for the product are stored (see 5.1).
- products* A list of **product_software_specs**.
Each **software_spec** shall refer to a product. Each **software_spec** shall be fully qualified. See 4.1.4.1 for the syntax of **software_spec**.

3.2 Distribution

A distribution contains product and bundle software objects. It is contained on a distribution media or may be part of the file store of a system. The distribution may contain a variety of software products and bundles, and that software may be applicable to a variety of hardware architectures or operating systems.

The distribution class inherits attributes from the `software_collection` common class.

A particular distribution object is identified within a host by the *path* attribute. For distributions, the *path* attribute is the pathname to the directory containing a distribution in the directory format of the software packaging layout, or a file or device file containing a distribution in a serial format of the software packaging layout.

Distributions can contain more than one version of a product or bundle. A version is uniquely identified within a distribution by the values of the *revision*, *vendor_tag*, and *architecture* attributes.

Table 3-2 – Attributes of the Distribution Class

Attribute	Length	Permitted Values	Default Value
<i>uuid</i>	64	Portable character string	Empty string
<i>media</i>	Undefined	List of media <i>sequence_number</i> values	Empty list

3.2.1 Distribution Attributes

The attributes listed in Table 3-2 and described in the following, along with the attributes listed in Table 3-1, characterize each instance of the distribution class:

media A list of *media.sequence_number* values for the distribution if the distribution spans multiple media. Each medium in a distribution shall have its *media.sequence_number* in the **INDEX** file defined for that medium. See 5.3. An implementation may include definitions for all media in the global **INDEX** file found on the first medium in the distribution. The *media.sequence_number* for the first medium in the distribution shall be 1 and shall be the first item in the list.

uuid A string that should uniquely identify a distribution.

The way in which a unique string is generated is undefined. This attribute is used for determining whether subsequent media are from the same set as the one that an install or copy started with. This attribute shall be defined for distributions that span multiple media.

3.3 Media

The media class is used to describe the media attributes for distributions that span multiple media.

Table 3-3 – Attributes of the Media Class

Attribute	Length	Permitted Values	Default Value
<i>sequence_number</i>	64	Portable character string	1

3.3.1 Media Attributes

The attribute listed in Table 3-3 and described in the following, characterizes each instance of the media class:

sequence_number

Identifies a particular media when a distribution spans multiple media.

It is used for identifying the correct medium on which to find the distribution files when the distribution spans multiple media.

3.4 Installed_Software

The installed_software class is used to describe the bundle and product software that has been installed on a file system.

The installed_software class inherits attributes from the software_collection common class.

A particular installed_software object is identified within a host by both the *path* attribute (defined in the software_collection class) and the *catalog* attribute. For installed_software objects, the *path* attribute is the root directory for the installed_software object below which all the software files are installed.

An installed_software object can contain multiple versions of a product or bundle. Multiple product and bundle versions are distinguished by the same attributes as distribution products, plus the user-specifiable *location* and *qualifier* attributes. Multiple product versions may be installed at the same time in an installed_software object. Different product versions may be installed into different locations, and different filesets from different product versions may be installed in the same location.

Table 3-4 – Attributes of the Installed Software Class

Attribute	Length	Permitted Values	Default Value
<i>catalog</i>	Undefined	Portable character string	Undefined

3.4.1 Installed_Software Attributes

The attribute listed in Table 3-4 and described in the following, along with the attributes listed in Table 3-1, characterizes each instance of the `installed_software` class:

catalog Along with the *path* attribute, identifies a single `installed_software` object.

Different `installed_software` objects may have the same value for the *path* attribute if and only if the value of their *catalog* attributes are different.

The *catalog* attribute is evaluated relative to the *path* attribute. It may be a POSIX.1 {2} pathname or other identifier: together they form the key to the undefined catalog storage for this `installed_software` object.

3.5 Vendor

The vendor class is used to describe the attributes of the vendors associated with products and bundles.

Each product or bundle identifies a vendor with a *vendor_tag* that identifies a particular vendor object. The *vendor_tag* attribute is used to distinguish products and bundles from different vendors that share the same product or bundle *tag*.

Table 3-5 – Attributes of the Vendor Class

Attribute	Length	Permitted Values	Default Value
<i>tag</i>	64	Filename character string	Empty string
<i>title</i>	256	Portable character string	Empty string
<i>description</i>	Undefined	Portable character string	Empty string

3.5.1 Vendor Attributes

The attribute listed in Table 3-5 and described in the following, characterizes each instance of the `installed_software` class:

<i>description</i>	A more detailed description of the vendor or information about the vendor.
<i>tag</i>	A short identifying name of the vendor that supplied the product. This attribute is used to to distinguish products and bundles from different vendors, and for resolving software specifications. Each software vendor should attempt to have a unique value for the <i>tag</i> attribute.
<i>title</i>	A longer name of the vendor that supplied the product. It is used for presentation purposes.

3.6 Software

Software is the common class from which products, bundles, filesets and subproducts inherit.

Table 3-6 – Attributes of the Software Common Class

Attribute	Length	Permitted Values	Default Value
<i>tag</i>	64	Filename character string	None
<i>create_time</i>	16	Integer character string	None
<i>description</i>	Undefined	Portable character string	Empty string
<i>mod_time</i>	16	Integer character string	None
<i>size</i>	32	Integer character string	None
<i>title</i>	256	Portable character string	Empty string

3.6.1 Software Common Attributes

The attributes listed in Table 3-6 and described in the following, characterize each instance of the software common class, and are inherited by each instance of the product, bundle, fileset, and subproduct classes:

<i>create_time</i>	A value that shall be set by the implementation to be the time that the catalog information for this object was first written. Time shall be represented as seconds since the Epoch, as defined in 5.6.1.3 of POSIX.1 {2}.
<i>description</i>	A more detailed description of the software object.
<i>mod_time</i>	A value that shall be set by the implementation to be the time that the catalog information for this object was last written.

251		Time shall be represented as seconds since the Epoch, as defined
252		in 5.6.1.3 of POSIX.1 {2}.
253	<i>size</i>	The sum of the sizes in bytes of all files and control_files con-
254		tained within the software object.
255		For objects other than filesets, the value is computed dynami-
256		cally as required. See 5.2.6, 5.2.7, and 5.2.8.
257	<i>tag</i>	A short name associated with the software object.
258		It is the one attribute that is always required to identify a
259		software object. For more information on software selections,
260		see 4.1.4.1.
261	<i>title</i>	A longer name associated with the software object, used for
262		display purposes.

263 3.7 Products

264 Products can contain filesets, which can be grouped into subproducts. Products
 265 are named by their *tag* attributes. A particular product object is uniquely
 266 identified within a software_collection by the *tag* attribute and by the version dis-
 267 tinguishing attributes. The attributes that uniquely distinguish a particular pro-
 268 duct version within a software_collection are *revision*, *architecture*, *vendor_tag*,
 269 *location*, and *qualifier*.

270 The product class shall inherit the attributes of the software common class.

271 See 4.1.4.1.2 on software compatibility.

272 3.7.1 Product Attributes

273 The product attributes listed in Table 3-7 and described in the following, along
 274 with the attributes listed in Table 3-6, characterize each instance of the product
 275 class:

276	<i>all_filesets</i>	This is a list of all filesets defined for the product, as opposed to
277		what is currently installed, described by the <i>filesets</i> attribute.
278		The <i>all_filesets</i> attribute is used to determine completeness of
279		this product when another software object has a dependency on
280		this product. In checking a product prerequisite or corequisite,
281		the existence of a <i>fileset.tag</i> in <i>all_filesets</i> that is not actually
282		installed or available indicates that the dependency is not
283		satisfied.

284		This does not affect exrequisites as they test whether any of the
285		contents of the dependency specification are present instead of
286		all of the contents tested for prerequisites or corequisites.

287	<i>architecture</i>	A vendor-defined string used to distinguish variations of a pro-
288		duct.

It is used for presentation purposes and for resolving software specifications. If a product with the same value of the *revision*

Table 3-7 – Attributes of the Product Class

Attribute	Length	Permitted Values	Default Value
<i>architecture</i>	64	Portable character string	Empty string
<i>location</i>	Undefined	Pathname character string	< <i>product.directory</i> >
<i>qualifier</i>	64	Portable character string	Empty String
<i>revision</i>	64	Portable character string	Empty string
<i>vendor_tag</i>	64	Filename character string	Empty string
<i>all_filesets</i>	Undefined	List of <i>fileset.tag</i> values	Empty list
<i>control_directory</i>	Undefined	Filename character string	< <i>product.tag</i> >
<i>copyright</i>	Undefined	Portable character string	Empty string
<i>directory</i>	Undefined	Pathname character string	/
<i>instance_id</i>	16	Filename character string	1
<i>is_locatable</i>	8	One of: true, false	true
<i>postkernel</i>	Undefined	Pathname character string	implementation defined
<i>layout_version</i>	64	1.0	1.0
<i>machine_type</i>	64	Software pattern matching string	Empty string
<i>number</i>	64	Portable character string	Empty string
<i>os_name</i>	64	Software pattern matching string	Empty string
<i>os_release</i>	64	Software pattern matching string	Empty string
<i>os_version</i>	64	Software pattern matching string	Empty string
<i>control_files</i>	Undefined	List of <i>control_file.tag</i> values	Empty list
<i>subproducts</i>	Undefined	List of <i>subproduct.tag</i> values	Empty list
<i>filesets</i>	Undefined	List of <i>fileset.tag</i> values	Empty list

and *vendor_tag* attributes has different versions of software for different target architectures, or any other variation (such as supported locale), then the value of the *architecture* attribute shall be different for each version. No additional semantics shall be assumed for its value.

control_directory

The name of the product control directory below which the *control_files* for the product are stored within an exported catalog.

See 5.1.

control_files

A list of the values of the *tag* attribute for all the *control_files* in the product.

These scripts are executed before and after software load, and before and after software removal.

copyright

The copyright notice for the product.

directory

The vendor-defined directory commonly associated with the product.

Generally, this will be the directory in or below which all (or mostly all) files within the product are installed.

For a product that has *filesets* with *is_locatable* equals **true**, all files that contain this directory as the first part of their path can

332		be relocated to the <i>location</i> directory during installation by
333		replacing the <i>product.directory</i> portion with the
334		<i>product.location</i> .
335	<i>filesets</i>	A list of the values of the <i>tag</i> attribute for all the filesets in the
336		product that are currently installed (in an <i>installed_software</i>
337		object) or available (in a distribution).
338	<i>instance_id</i>	A single attribute that distinguishes versions of products (and
339		bundles) with the same tag.
340		It is a simple form of the version distinguishing attributes, valid
341		only within the context of an exported catalog.
342	<i>is_locatable</i>	A Boolean value indicating whether any of the filesets in the pro-
343		duct have the <i>is_locatable</i> attribute set to true .
344	<i>layout_version</i>	
345		This attribute, and its value, are included for future use.
346	<i>location</i>	Used for resolving software_specs for installed software.
347		A specific product location refers to all filesets of that product
348		that are installed at that location. This is the path beneath
349		which the relocatable files of that product are stored. See
350		4.5.7.3.1.
351		This attribute is valid only for products in <i>installed_software</i> .
352	<i>machine_type</i>	
353		A software pattern matching string describing valid machine
354		members of the <i>uname</i> structure as defined by 4.4.1 of
355		POSIX.1 {2}.
356		It is used for determining compatibility.
357	<i>number</i>	The semantics associated with the values of this attribute are
358		undefined.
359		This attribute can be used to store such vendor-defined values as
360		part number, order number or serial number.
361	<i>os_name</i>	A software pattern matching string describing valid sysname
362		members of the <i>uname</i> structure as defined by 4.4.1 of
363		POSIX.1 {2}.
364		It is used for determining compatibility.
365	<i>os_release</i>	A software pattern matching string describing valid release
366		members of the <i>uname</i> structure as defined by 4.4.1 of
367		POSIX.1 {2}.
368		It is used for determining compatibility.
369	<i>os_version</i>	A software pattern matching string describing valid version
370		members of the <i>uname</i> structure as defined by 4.4.1 of
371		POSIX.1 {2}.
		It is used for determining compatibility.

372	<i>postkernel</i>	
373		The path to the script that is run after the kernel filesets have
374		been installed.
375		Any product containing kernel filesets should include this path.
376		If this attribute is supplied, the corresponding script shall be run
377		if it exists relative to the root directory of the <i>installed_software</i> .
378		If this attribute is not supplied, then the implementation-defined
379		path (the default value for the attribute) shall be used if it exists
380		relative to the root directory of the <i>installed_software</i> . Note that
381		the use of an alternate root directory may mean that the default
382		path does not exist relative to the root directory of the
383		<i>installed_software</i> .
384	<i>qualifier</i>	Specified by a user when installing software and used for
385		identifying a product (or set of product versions) using a logical
386		name.
387		Applies only to products in <i>installed_software</i> .
388	<i>revision</i>	A vendor-defined string describing the revision of the product.
389		It is used for presentation purposes and for resolving software
390		specifications. The revision shall be interpreted as a . (period)
391		separated string. See 4.1.4.1.
392	<i>subproducts</i>	A list of the values of the <i>tag</i> attribute for all the subproducts in
393		the product.
394	<i>vendor_tag</i>	A short identifying name of the vendor that supplied the
395		product.
396		This attribute may also be used to identify a vendor object
397		containing additional attributes describing the vendor.
398		This attribute is used to distinguish software objects, allowing
399		more than one vendor to produce a product with the same value
400		of the other version distinguishing attributes. It is used for
401		presentation purposes and for resolving software specifications.

402 3.8 Bundles

403 Bundles are groupings of software objects. Bundles contain references to pro-
 404 ducts, parts of products, or other bundles. A software object can be referenced by
 405 more than one bundle.

406 The bundle class shall inherit the attributes of the software common class.

407 A particular bundle object is uniquely identified within a *software_collection* by
 408 the *tag* and by the version distinguishing attributes. The attributes that uniquely
 409 distinguish a particular bundle version are *revision*, *architecture*, *location*,
 410 *vendor_tag*, and *qualifier*.

411 Bundles, like products, are named by their *tag* attributes and share the same
 name space as products. Products and bundles shall be considered together in
 determining a unique value for *instance_id*.

3.8 Bundles

Bundles and products include many of the same attributes. No bundle attributes are automatically derived from the contained product attributes. They are defined independently. See 4.1.4.1.2 on software compatibility.

Bundle definitions are copied or installed when explicitly specified in a software selection for **swcopy** and **swinstall** respectively. They remain installed until explicitly removed or until all of their contents are removed.

Table 3-8 – Attributes of the Bundle Class

Attribute	Length	Permitted Values	Default Value
<i>architecture</i>	64	Portable character string	Empty string
<i>location</i>	Undefined	Pathname character string	< <i>bundle.directory</i> >
<i>qualifier</i>	64	Portable character string	Empty String
<i>revision</i>	64	Portable character string	Empty string
<i>vendor_tag</i>	64	Filename character string	Empty string
<i>contents</i>	Undefined	List of software_specs	Empty list
<i>copyright</i>	Undefined	Portable character string	Empty string
<i>directory</i>	Undefined	Pathname character string	Empty string
<i>instance_id</i>	16	Filename character string	1
<i>is_locatable</i>	8	One of: true , false	true
<i>layout_version</i>	64	1.0	1.0
<i>machine_type</i>	64	Software pattern matching string	Empty string
<i>number</i>	64	Portable character string	Empty string
<i>os_name</i>	64	Software pattern matching string	Empty string
<i>os_release</i>	64	Software pattern matching string	Empty string
<i>os_version</i>	64	Software pattern matching string	Empty string

3.8.1 Bundle Attributes

The attributes listed in Table 3-8 and described in the following, along with the attributes listed in Table 3-6, characterize each instance of the bundle class:

- architecture* A vendor-defined string used to distinguish variations of a bundle.
It is used for presentation purposes and for resolving software specifications.
- contents* A list of **software_specs** that defines the list of software grouped into this bundle, as originally defined in the PSF.
- copyright* A copyright notice for the bundle.
- directory* The default directory (and location) of the bundle.
This is the default path prefixed, when the bundle is installed, to the location of each product and bundle specification within this bundle.
- instance_id* A single attribute that distinguishes versions of bundles (and products) with the same tag.

453		It is a simple form of the version distinguishing attributes, valid
454		only within the context of an exported catalog.
455	<i>is_locatable</i>	A Boolean value indicating whether any of the contents in the
456		bundle have the <i>is_locatable</i> attribute set to true .
457	<i>layout_version</i>	
458		This attribute, and its value, are included for future use.
459	<i>location</i>	An attribute whose value is set when installing software and
460		used for resolving software_specs for installed software.
461		When installing a bundle the <i>bundle.location</i> is prefixed to the
462		location specification for each software_spec in the contents
463		of the bundle, before that software_spec is resolved.
464		The <i>contents</i> attribute of the bundle is not modified.
465		Applies only to bundles in <i>installed_software</i> .
466	<i>machine_type</i>	A software pattern matching string describing valid machine
467		members of the <i>uname</i> structure as defined in 4.4.1 of
468		POSIX.1 {2}.
469		It is used for determining compatibility.
470	<i>number</i>	The semantics associated with the values of this attribute are
471		undefined.
472		This attribute can be used to store such vendor-defined values as
473		part number, order number or serial number.
474	<i>os_name</i>	A software pattern matching string describing valid sysname
475		members of the <i>uname</i> structure as defined in 4.4.1 of
476		POSIX.1 {2}.
477		It is used for determining compatibility.
478	<i>os_release</i>	A software pattern matching string describing valid release
479		members of the <i>uname</i> structure as defined in 4.4.1 of
480		POSIX.1 {2}.
481		It is used for determining compatibility.
482	<i>os_version</i>	A software pattern matching string describing valid version
483		members of the <i>uname</i> structure as defined in 4.4.1 of
484		POSIX.1 {2}.
485		It is used for determining compatibility.
486	<i>qualifier</i>	Specified by a user when installing software, and used for identi-
487		fying a bundle (or set of bundle versions) using a logical name.
488		Applies only to bundles in <i>installed_software</i> .
489	<i>revision</i>	A vendor-defined string used to distinguish different revisions of
490		bundles from one another.
491		It is used for presentation purposes and for resolving software
		specifications.

3.8 Bundles